EXHIBIT A

US005796967A

United States Patent [19]

Filepp et al.

[11] Patent Number: 5,796,967

[45] Date of Patent: Aug. 18, 1998

[54] **METHOD FOR PRESENTING APPLICATIONS IN AN INTERACTIVE SERVICE**

[75] Inventors: **Robert Filepp**, Springfield, N.J.; **Kenneth H. Appleman**, White Plains; **Alexander W. Bidwell**, New York, both of N.Y.; **Allan M. Wolf**, Ridgefield; **James A. Galambos**, Westport, both of Conn.; **Sam Meo**, New York, N.Y.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **158,031**

[22] Filed: **Nov. 26, 1993**

Related U.S. Application Data

[60] Division of Ser. No. 388,156, Jul. 28, 1989, Pat. No. 5,347,632, which is a continuation-in-part of Ser. No. 328,790, Mar. 23, 1989, abandoned, which is a continuation-in-part of Ser. No. 219,931, Jul. 15, 1988, abandoned.

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **395/339**; 395/200.08; 395/200.09

[58] **Field of Search** 395/155–159, 395/161, 160, 200.03, 200.04, 200.05, 200.08, 601, 326, 329–335, 339–349, 352–354, 356–357

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,821,211	4/1989	Torres	395/357
4,949,248	8/1990	Caro	395/200.09
4,953,159	8/1990	Hayden et al.	395/200.04
5,050,105	9/1991	Peters	395/346
5,220,657	6/1993	Bly et al.	395/153
5,280,583	1/1994	Nakayama et al.	395/153

OTHER PUBLICATIONS

McGregor, "Designing User Interface Tools for the X Window System", Compcon Spring '89 IEEE Computer Society Int'l Conference, pp. 243–246., 1989.

Dunwoody et al, "A Dynamic Profile of Window System Usage", Computer Workstations Conference, pp. 90–99., 1988.

Gancarz, "Uwm: A User Interface for X Windows", Usenix Association Summer Conference Proceedings, 1986.

Scheifler et al, "The X Windows System", ACM Transactions on Graphics, vol. 5, No. 2, pp. 79–709., Apr. 1986.

Microsoft Windows version 2.0 user's guide, pp. 88–91., 1987.

"Microsoft Windows—Version 2.0", 1987, (Book 1; p. viii–xiii; Book 4, pp. 36.37.83).

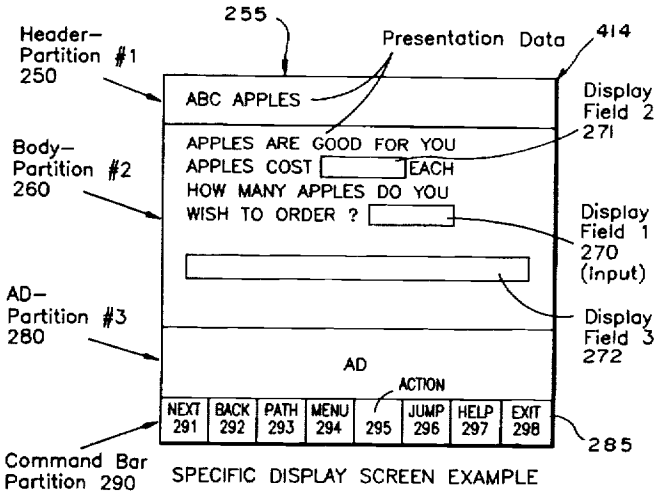
Mastering Windows™ 3.0 by Robert Cowart, 1990, pp. 6–27.

Primary Examiner—Joseph H. Field
Attorney, Agent, or Firm—Paul C. Scifo

[57] **ABSTRACT**

A method for presenting applications in an interactive service featuring steps for generating screen displays of the service applications at the reception systems of the respective users. Steps are provided for generating the application displays as screens having a plurality of partitions, the partitions being constructed from reusable elements. In accord with the method, the screens include at least a first partition at which an application may be presented and a second, concurrently displayed partition including command functions for managing the display. The method further includes steps for providing command functions that facilitate random navigation to new applications with a variety of different procedures which the user can choose from. In (preferred) one form, the functions are presented as a command bar located at the bottom of the screen. Further, the method includes steps for opening and closing windows on the display to enable presentation of additional data relating to the presented applications. Still further, the method includes steps for providing additional partitions for concurrently displaying other applications, which may include advertising.

17 Claims, 16 Drawing Sheets



SPECIFIC DISPLAY SCREEN EXAMPLE

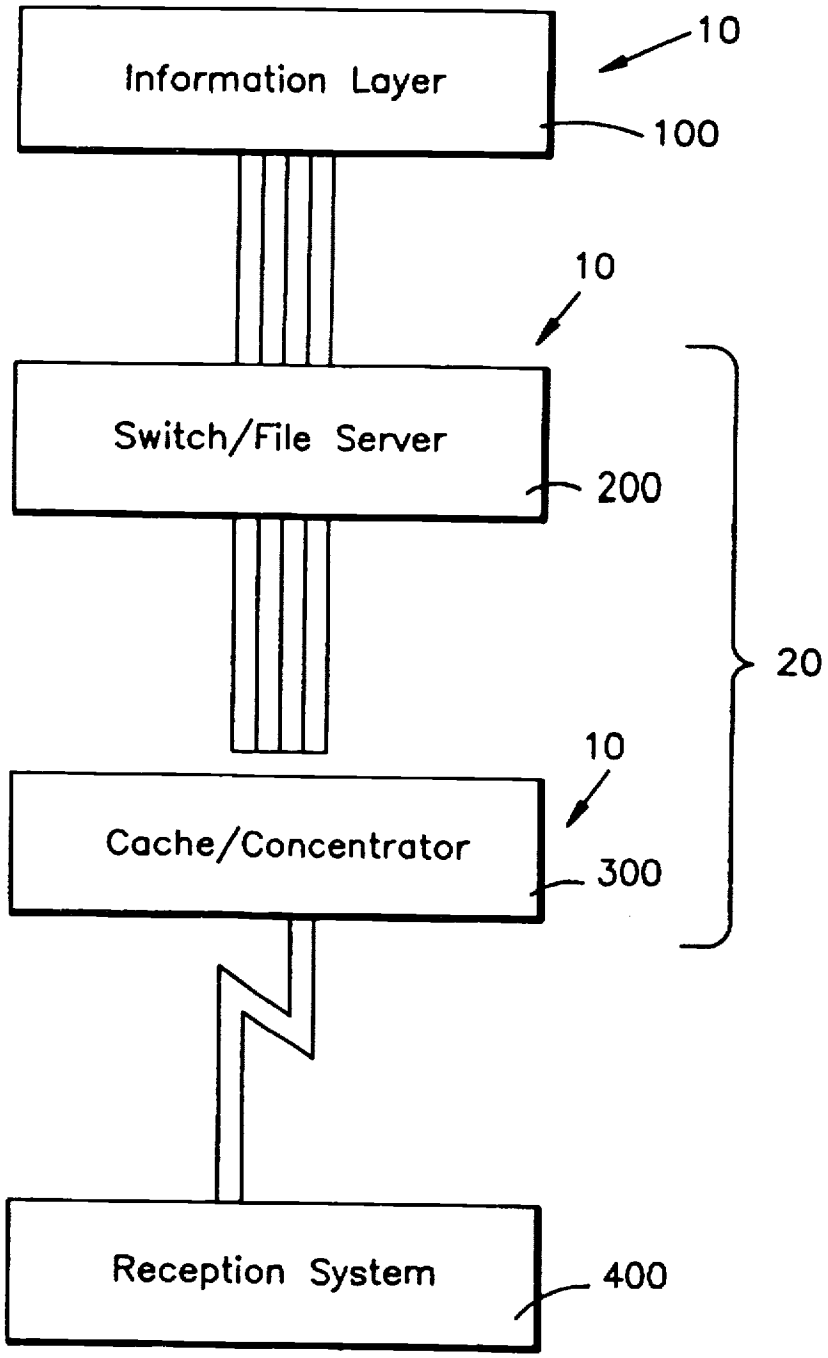

Information Layer
10
100
10
Switch/File Server
200
20
10
Cache/Concentrator
300
Reception System
400

FIG. 1

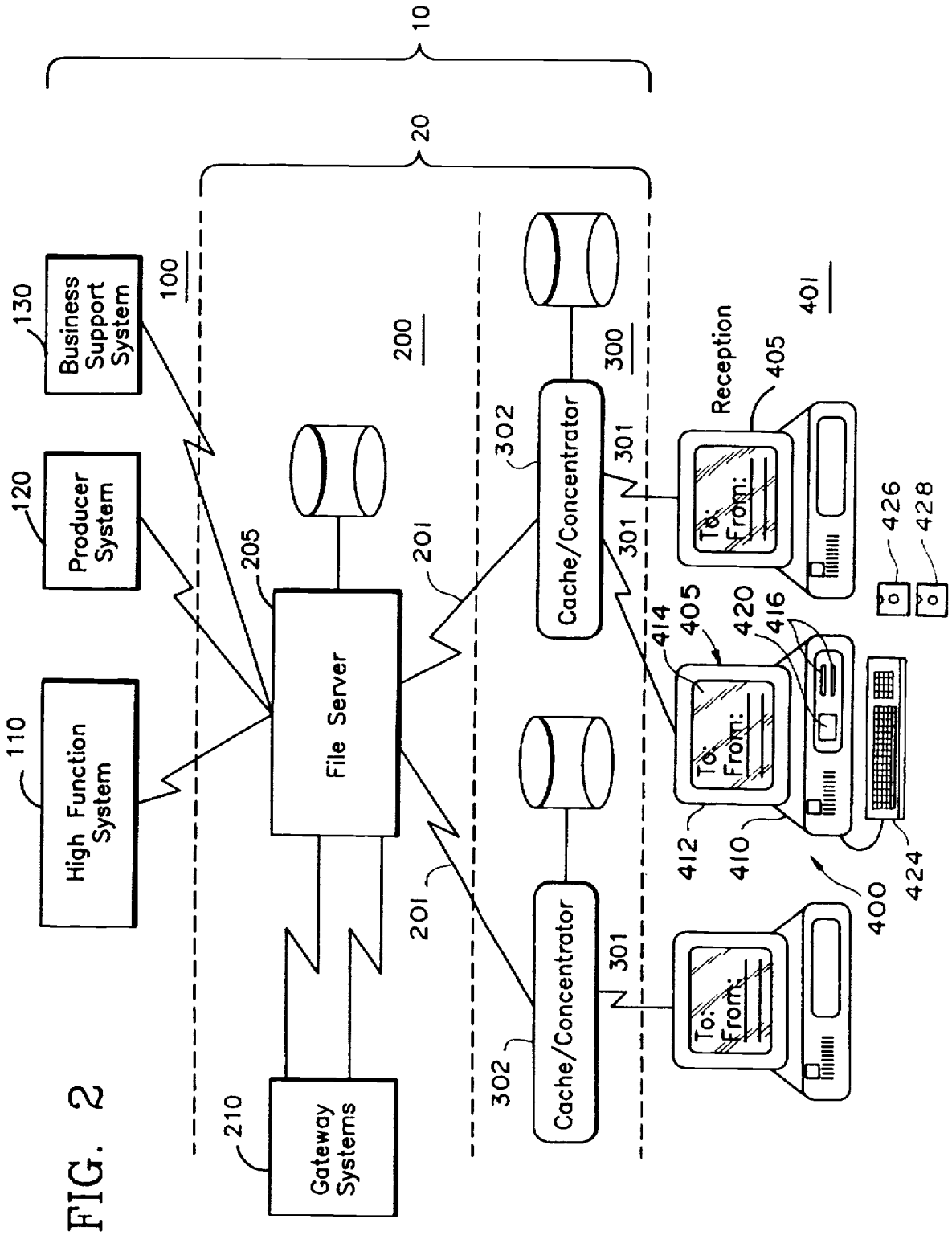

10
20
100
130
Business Support System
120
Producer System
110
High Function System
200
205
File Server
201
201
210
Gateway Systems
300
302
Cache/Concentrator
301
301
302
Cache/Concentrator
301
401
Reception
405
To:
From:
426
428
414
405
420
416
To:
From:
412
410
424
400
To:
From:

FIG. 2

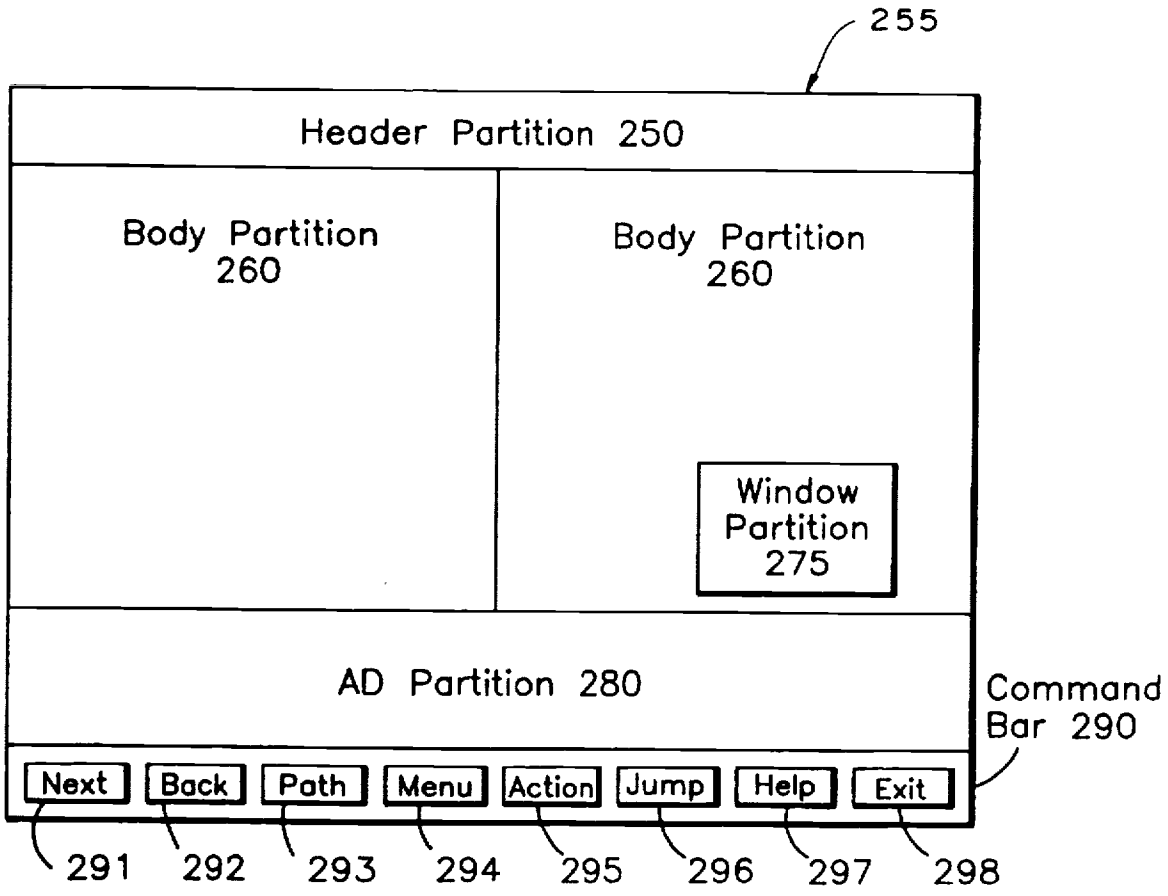

255
Header Partition 250
Body Partition 260
Body Partition 260
Window Partition 275
AD Partition 280
Command Bar 290
Next
Back
Path
Menu
Action
Jump
Help
Exit
291
292
293
294
295
296
297
298

FIG. 3a

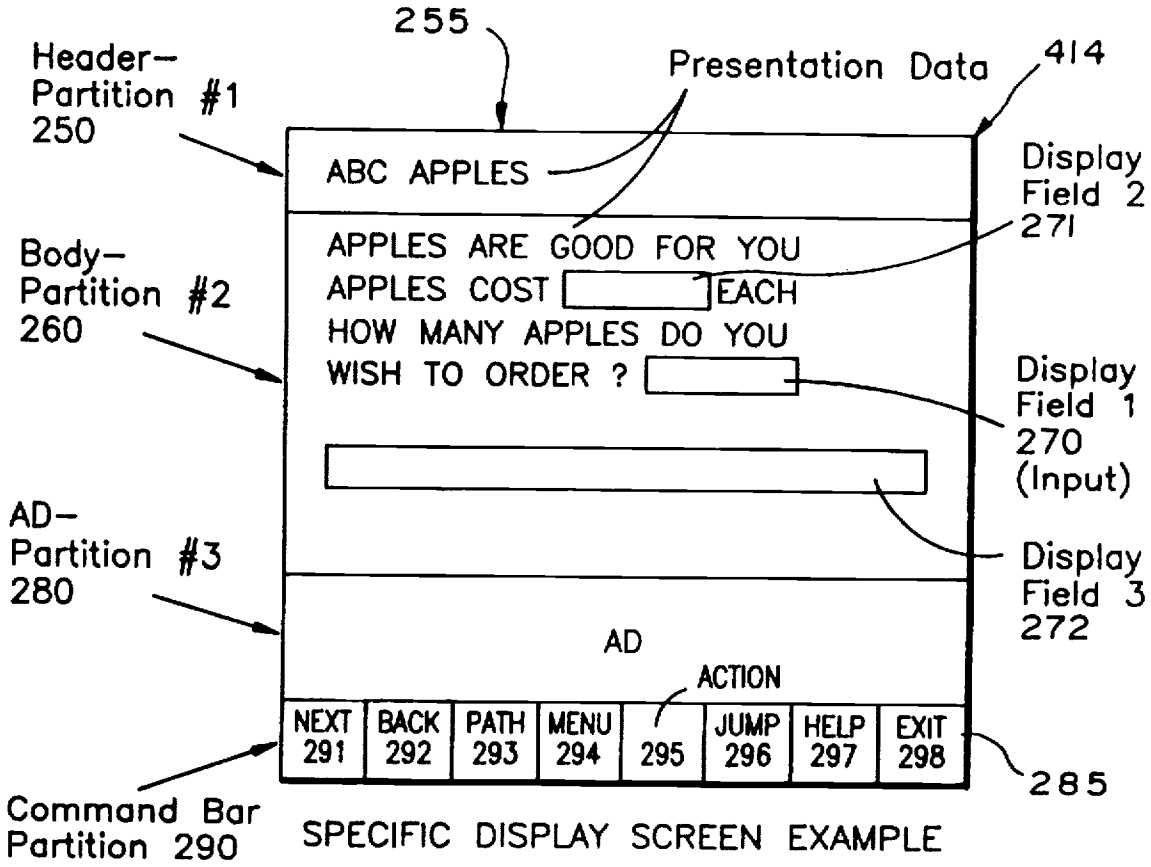

255
Header–
Partition #1
250
Presentation Data
414
ABC APPLES
Display
Field 2
271
APPLES ARE GOOD FOR YOU
APPLES COST EACH
HOW MANY APPLES DO YOU
WISH TO ORDER ?
Body–
Partition #2
260
Display
Field 1
270
(Input)
AD–
Partition #3
280
Display
Field 3
272
AD
ACTION
NEXT 291
BACK 292
PATH 293
MENU 294
295
JUMP 296
HELP 297
EXIT 298
285
Command Bar
Partition 290
SPECIFIC DISPLAY SCREEN EXAMPLE

FIG. 3b

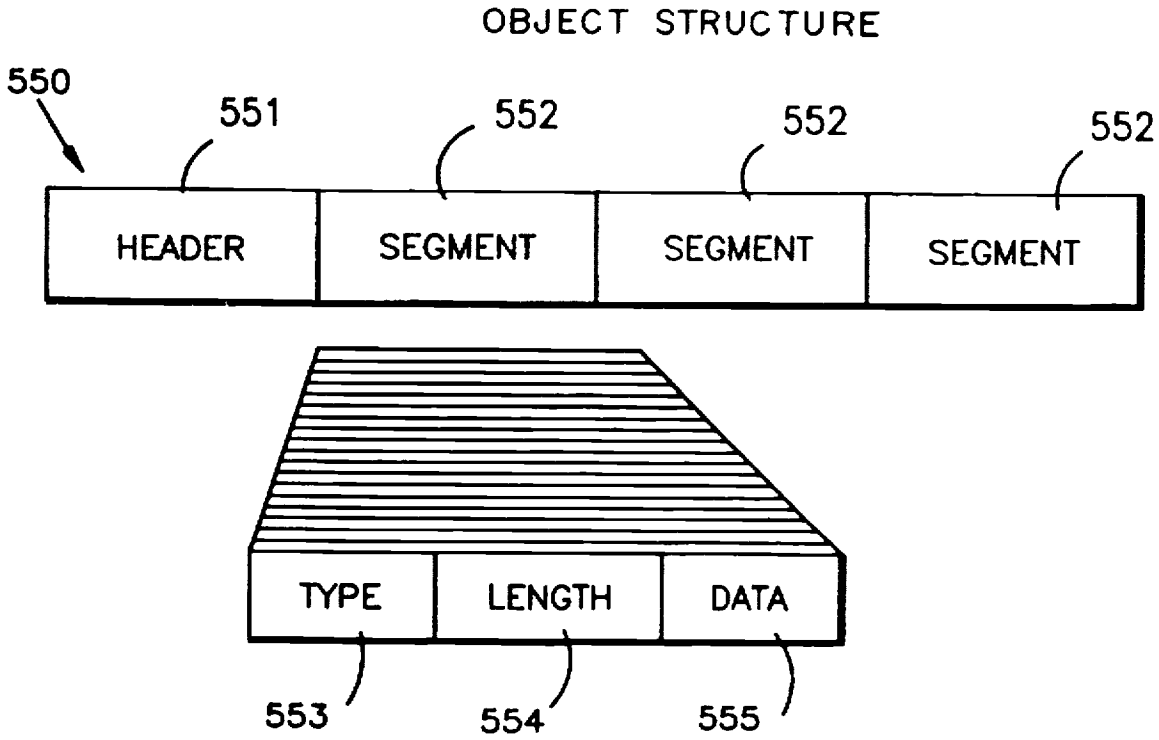

OBJECT STRUCTURE
550
551
552
552
552
HEADER
SEGMENT
SEGMENT
SEGMENT
TYPE
LENGTH
DATA
553
554
555

FIG. 4a

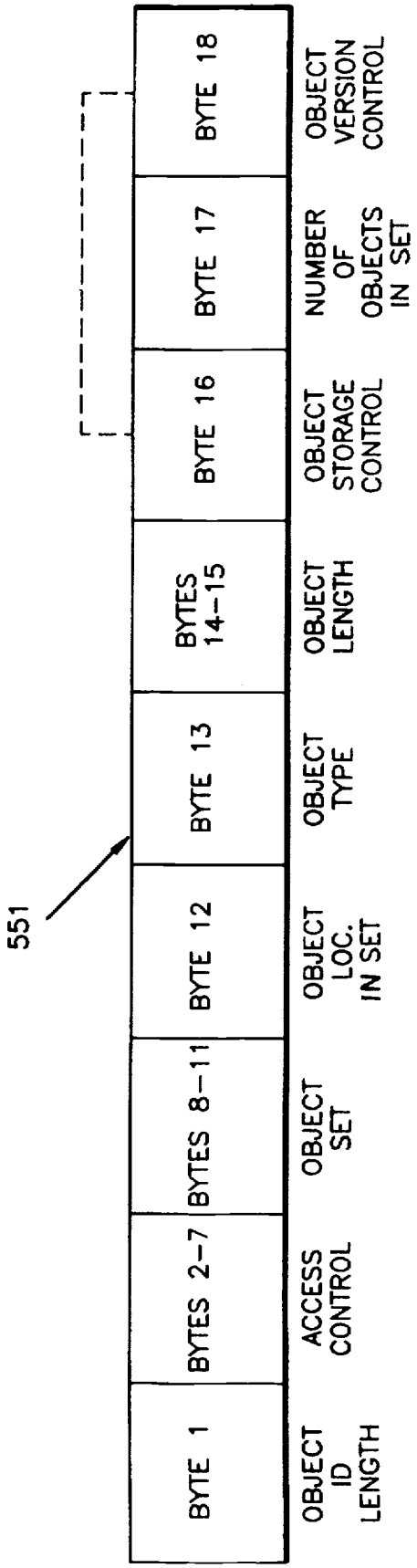

551
BYTE 1
BYTES 2–7
BYTES 8–11
BYTE 12
BYTE 13
BYTES 14–15
BYTE 16
BYTE 17
BYTE 18
OBJECT ID LENGTH
ACCESS CONTROL
OBJECT SET
OBJECT LOC. IN SET
OBJECT TYPE
OBJECT LENGTH
OBJECT STORAGE CONTROL
NUMBER OF OBJECTS IN SET
OBJECT VERSION CONTROL

FIG. 4b

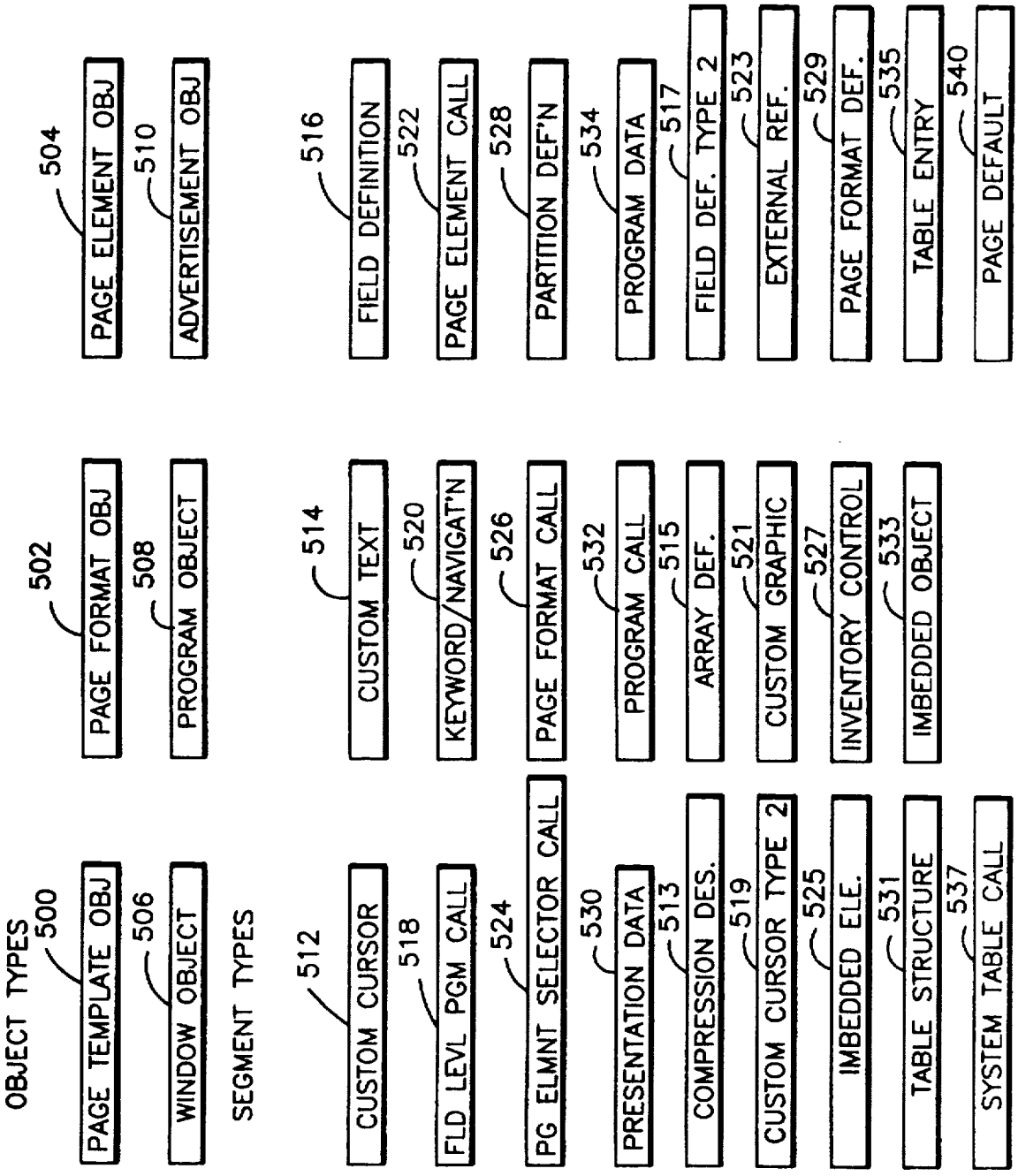

OBJECT TYPES
500
PAGE TEMPLATE OBJ
506
WINDOW OBJECT
502
PAGE FORMAT OBJ
508
PROGRAM OBJECT
504
PAGE ELEMENT OBJ
510
ADVERTISEMENT OBJ
SEGMENT TYPES
512
CUSTOM CURSOR
518
FLD LEVL PGM CALL
524
PG ELMNT SELECTOR CALL
530
PRESENTATION DATA
513
COMPRESSION DES.
519
CUSTOM CURSOR TYPE 2
525
IMBEDDED ELE.
531
TABLE STRUCTURE
537
SYSTEM TABLE CALL
514
CUSTOM TEXT
520
KEYWORD/NAVIGAT'N
526
PAGE FORMAT CALL
532
PROGRAM CALL
515
ARRAY DEF.
521
CUSTOM GRAPHIC
527
INVENTORY CONTROL
533
IMBEDDED OBJECT
516
FIELD DEFINITION
522
PAGE ELEMENT CALL
528
PARTITION DEF'N
534
PROGRAM DATA
517
FIELD DEF. TYPE 2
523
EXTERNAL REF.
529
PAGE FORMAT DEF.
535
TABLE ENTRY
540
PAGE DEFAULT

FIG. 4c

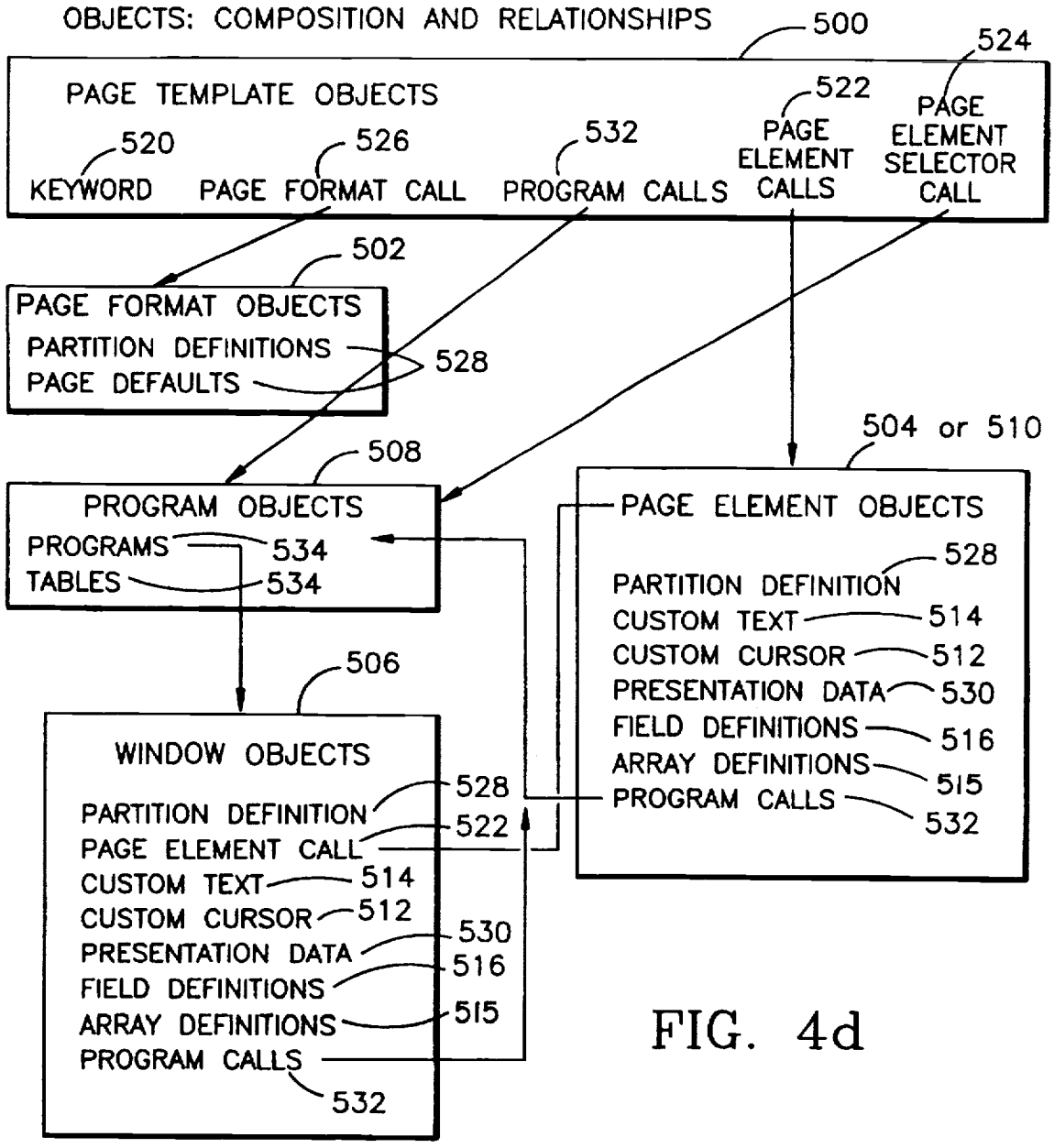

OBJECTS: COMPOSITION AND RELATIONSHIPS
500
PAGE TEMPLATE OBJECTS
524
522
520
526
532
KEYWORD
PAGE FORMAT CALL
PROGRAM CALLS
PAGE ELEMENT CALLS
PAGE ELEMENT SELECTOR CALL
502
PAGE FORMAT OBJECTS
PARTITION DEFINITIONS
PAGE DEFAULTS
528
504 or 510
508
PROGRAM OBJECTS
PROGRAMS
TABLES
534
534
PAGE ELEMENT OBJECTS
528
PARTITION DEFINITION
CUSTOM TEXT
514
CUSTOM CURSOR
512
PRESENTATION DATA
530
FIELD DEFINITIONS
516
ARRAY DEFINITIONS
515
PROGRAM CALLS
532
506
WINDOW OBJECTS
528
PARTITION DEFINITION
522
PAGE ELEMENT CALL
514
CUSTOM TEXT
512
CUSTOM CURSOR
530
PRESENTATION DATA
516
FIELD DEFINITIONS
515
ARRAY DEFINITIONS
PROGRAM CALLS
532

FIG. 4d

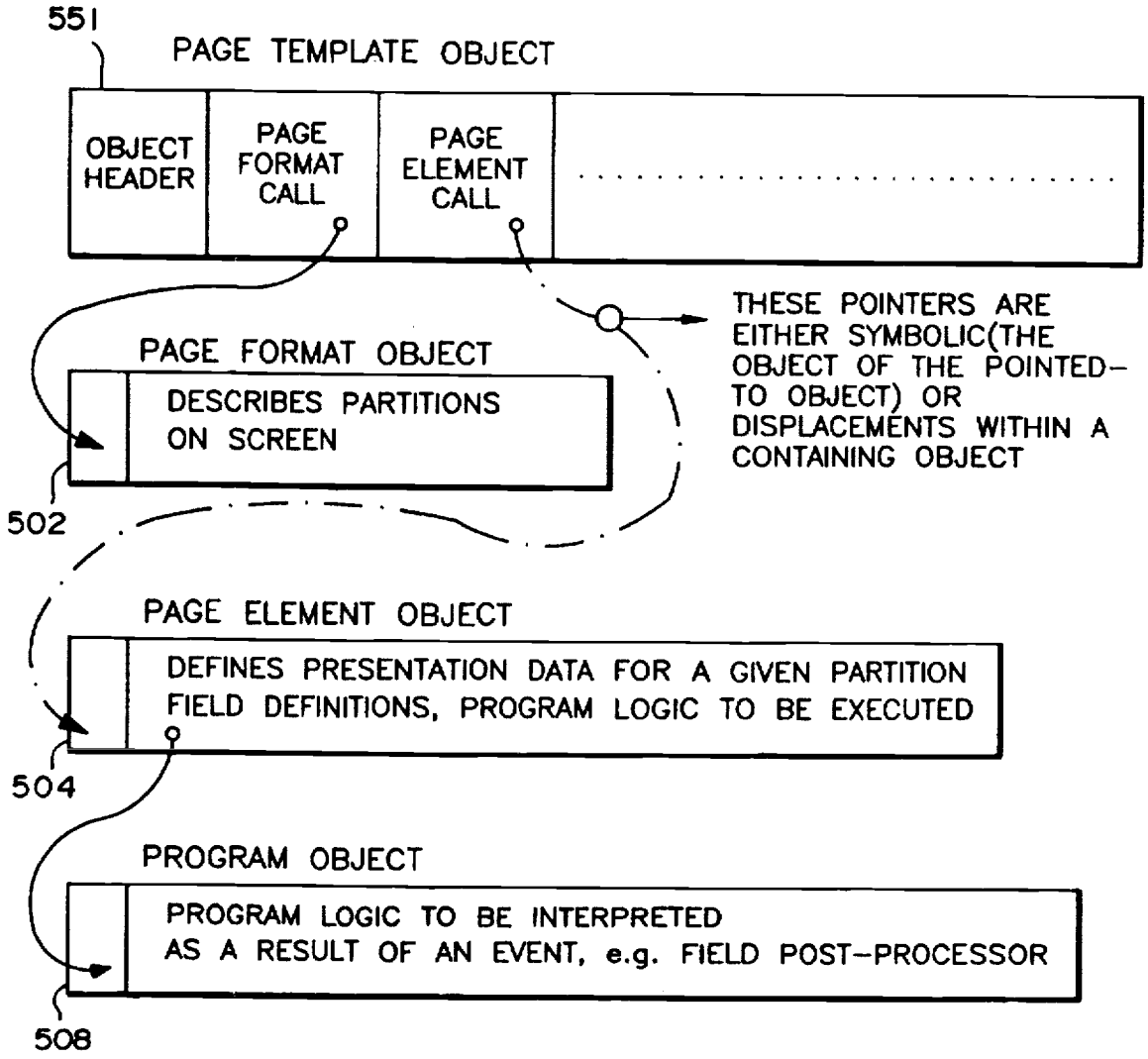

551
PAGE TEMPLATE OBJECT
OBJECT HEADER
PAGE FORMAT CALL
PAGE ELEMENT CALL
THESE POINTERS ARE EITHER SYMBOLIC(THE OBJECT OF THE POINTED-TO OBJECT) OR DISPLACEMENTS WITHIN A CONTAINING OBJECT
PAGE FORMAT OBJECT
DESCRIBES PARTITIONS ON SCREEN
502
PAGE ELEMENT OBJECT
DEFINES PRESENTATION DATA FOR A GIVEN PARTITION FIELD DEFINITIONS, PROGRAM LOGIC TO BE EXECUTED
504
PROGRAM OBJECT
PROGRAM LOGIC TO BE INTERPRETED AS A RESULT OF AN EVENT, e.g. FIELD POST-PROCESSOR
508

FIG. 5a

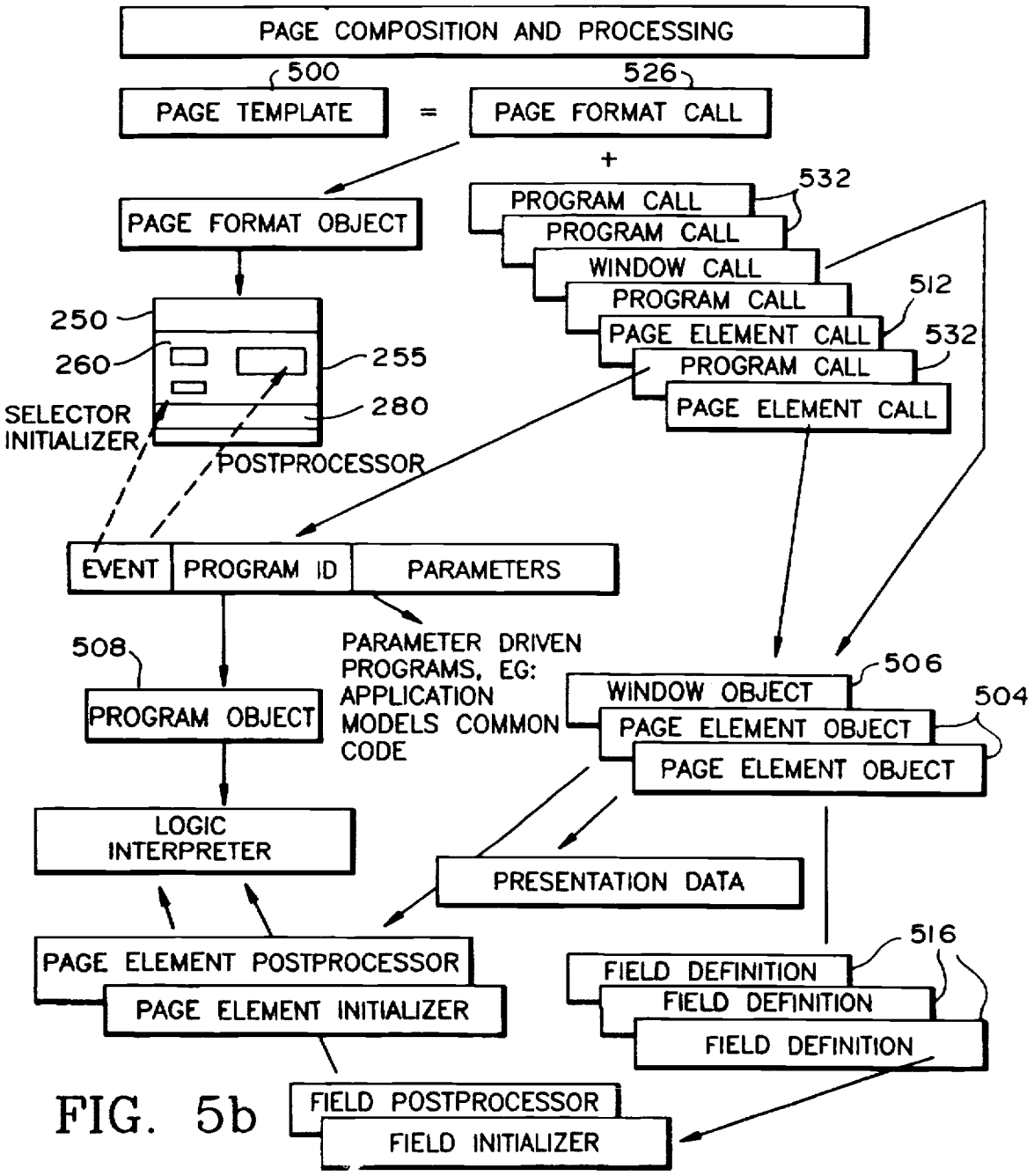
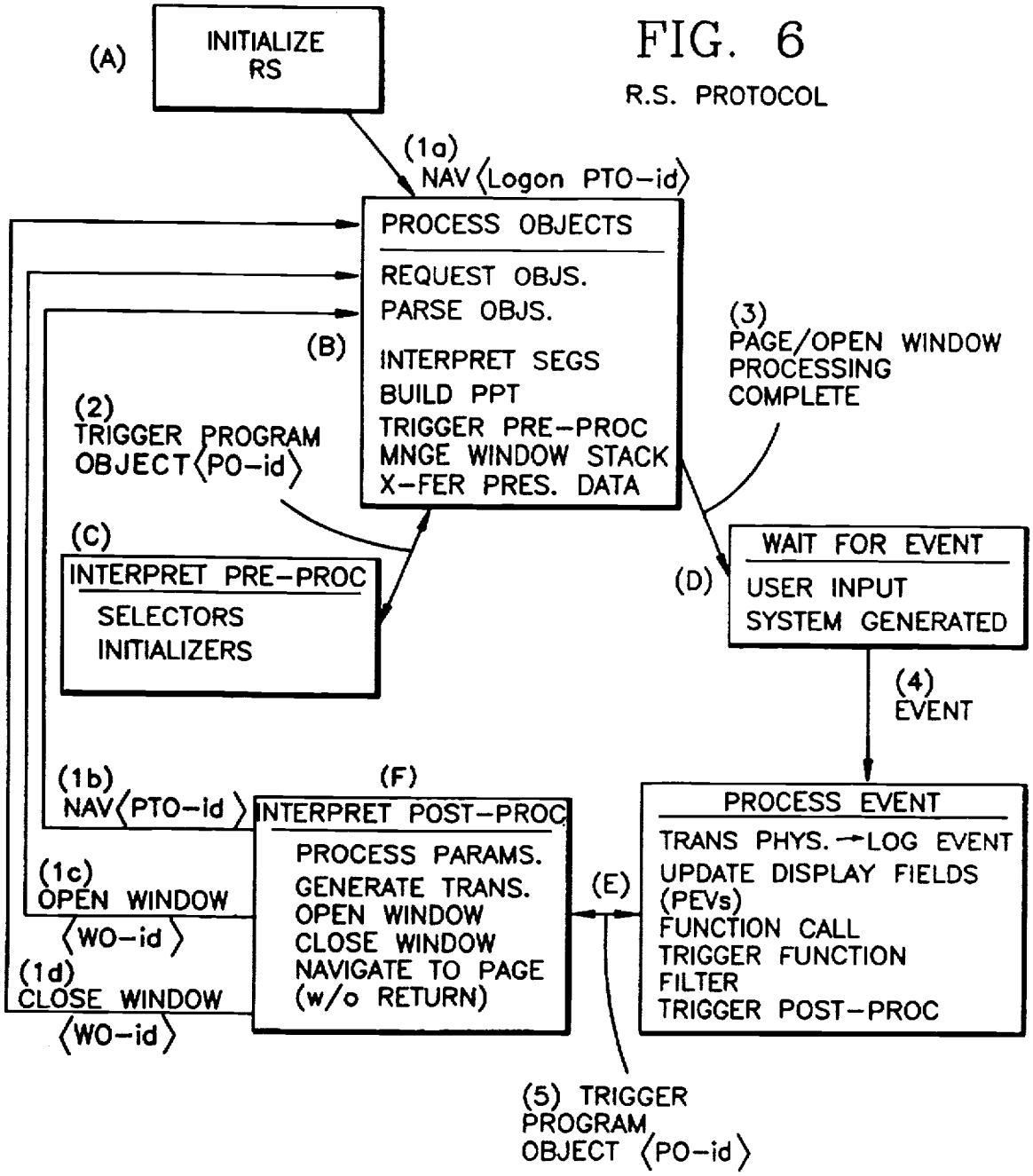
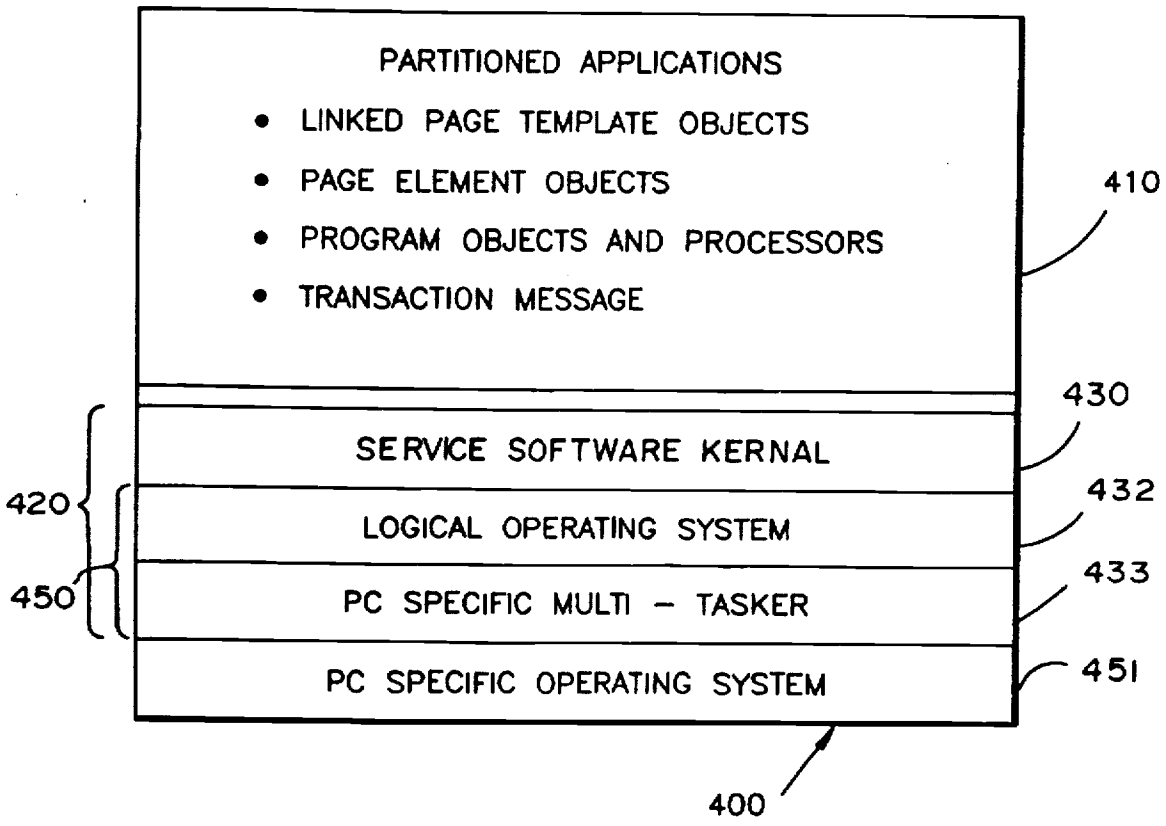

PAGE COMPOSITION AND PROCESSING
500
PAGE TEMPLATE
=
526
PAGE FORMAT CALL
+
PAGE FORMAT OBJECT
PROGRAM CALL
532
PROGRAM CALL
WINDOW CALL
PROGRAM CALL
512
PAGE ELEMENT CALL
532
PROGRAM CALL
PAGE ELEMENT CALL
250
260
255
280
SELECTOR INITIALIZER
POSTPROCESSOR
EVENT
PROGRAM ID
PARAMETERS
508
PROGRAM OBJECT
PARAMETER DRIVEN PROGRAMS, EG: APPLICATION MODELS COMMON CODE
506
WINDOW OBJECT
504
PAGE ELEMENT OBJECT
PAGE ELEMENT OBJECT
LOGIC INTERPRETER
PRESENTATION DATA
516
PAGE ELEMENT POSTPROCESSOR
PAGE ELEMENT INITIALIZER
FIELD DEFINITION
FIELD DEFINITION
FIELD DEFINITION
FIELD POSTPROCESSOR
FIELD INITIALIZER

FIG. 5b

FIG. 6

R.S. PROTOCOL





RECEPTION SYSTEM LAYERS

FIG. 7

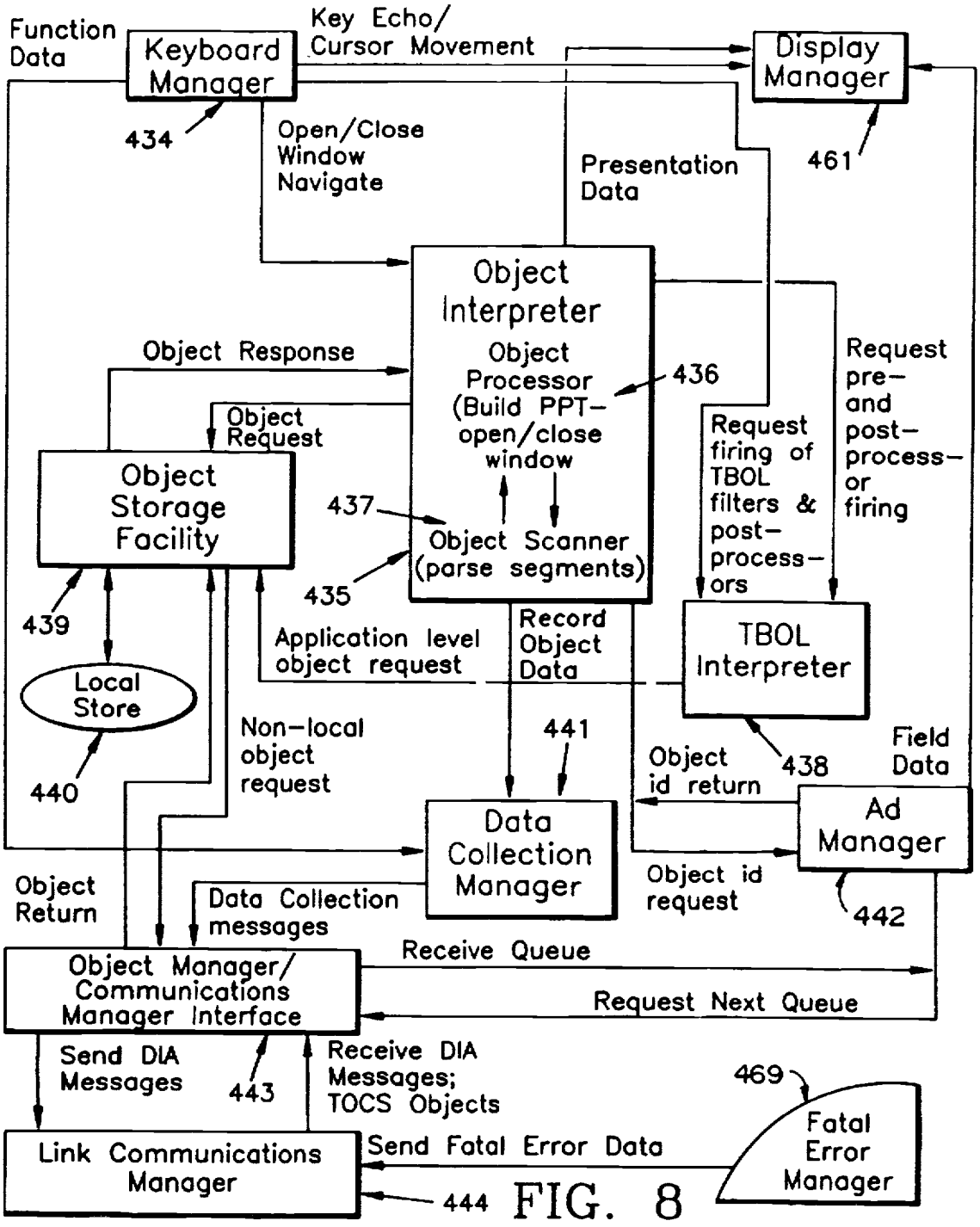

Function Data
Keyboard Manager
434
Key Echo/ Cursor Movement
Display Manager
461
Open/Close Window Navigate
Presentation Data
Object Interpreter
Object Processor (Build PPT- open/close window
436
Object Response
Object Request
Object Storage Facility
437
Object Scanner (parse segments)
435
Request pre- and post- process- or firing
Request firing of TBOL filters & post- process- ors
439
Application level object request
Record Object Data
TBOL Interpreter
Local Store
440
Non-local object request
441
Object id return
438
Field Data
Ad Manager
Data Collection Manager
Object Return
Data Collection messages
Object id request
442
Receive Queue
Object Manager/ Communications Manager Interface
Request Next Queue
Send DIA Messages
443
Receive DIA Messages; TOCS Objects
469
Fatal Error Manager
Send Fatal Error Data
Link Communications Manager
444

FIG. 8

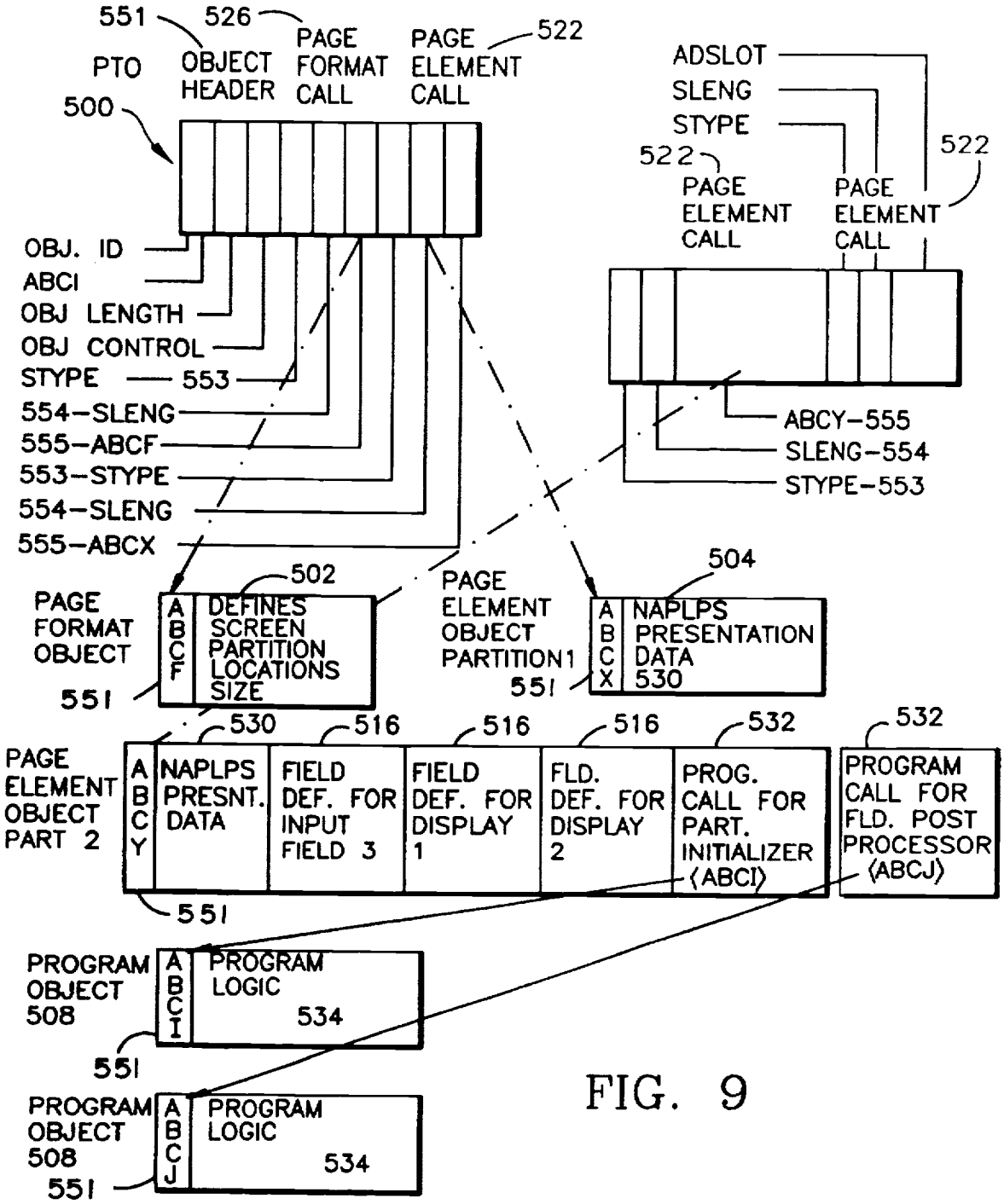

551
526
PAGE FORMAT CALL
PAGE ELEMENT CALL
522
PTO
OBJECT HEADER
500
ADSLOT
SLENG
STYPE
522
PAGE ELEMENT CALL
PAGE ELEMENT CALL
522
OBJ. ID
ABCI
OBJ LENGTH
OBJ CONTROL
STYPE
553
554–SLENG
555–ABCF
553–STYPE
554–SLENG
555–ABCX
ABCY–555
SLENG–554
STYPE–553
504
502
PAGE FORMAT OBJECT
A DEFINES
B SCREEN
C PARTITION
F LOCATIONS
SIZE
551
PAGE ELEMENT OBJECT PARTITION1
551
A NAPLPS
B PRESENTATION
C DATA
X 530
530
516
516
516
532
532
PAGE ELEMENT OBJECT PART 2
A
B
C
Y
NAPLPS PRESNT. DATA
FIELD DEF. FOR INPUT FIELD 3
FIELD DEF. FOR DISPLAY 1
FLD. DEF. FOR DISPLAY 2
PROG. CALL FOR PART. INITIALIZER ⟨ABCI⟩
PROGRAM CALL FOR FLD. POST PROCESSOR ⟨ABCJ⟩
551
PROGRAM OBJECT 508
A
B
C
I
PROGRAM LOGIC
534
551
PROGRAM OBJECT 508
A
B
C
J
PROGRAM LOGIC
534
551

FIG. 9

FIG. 10

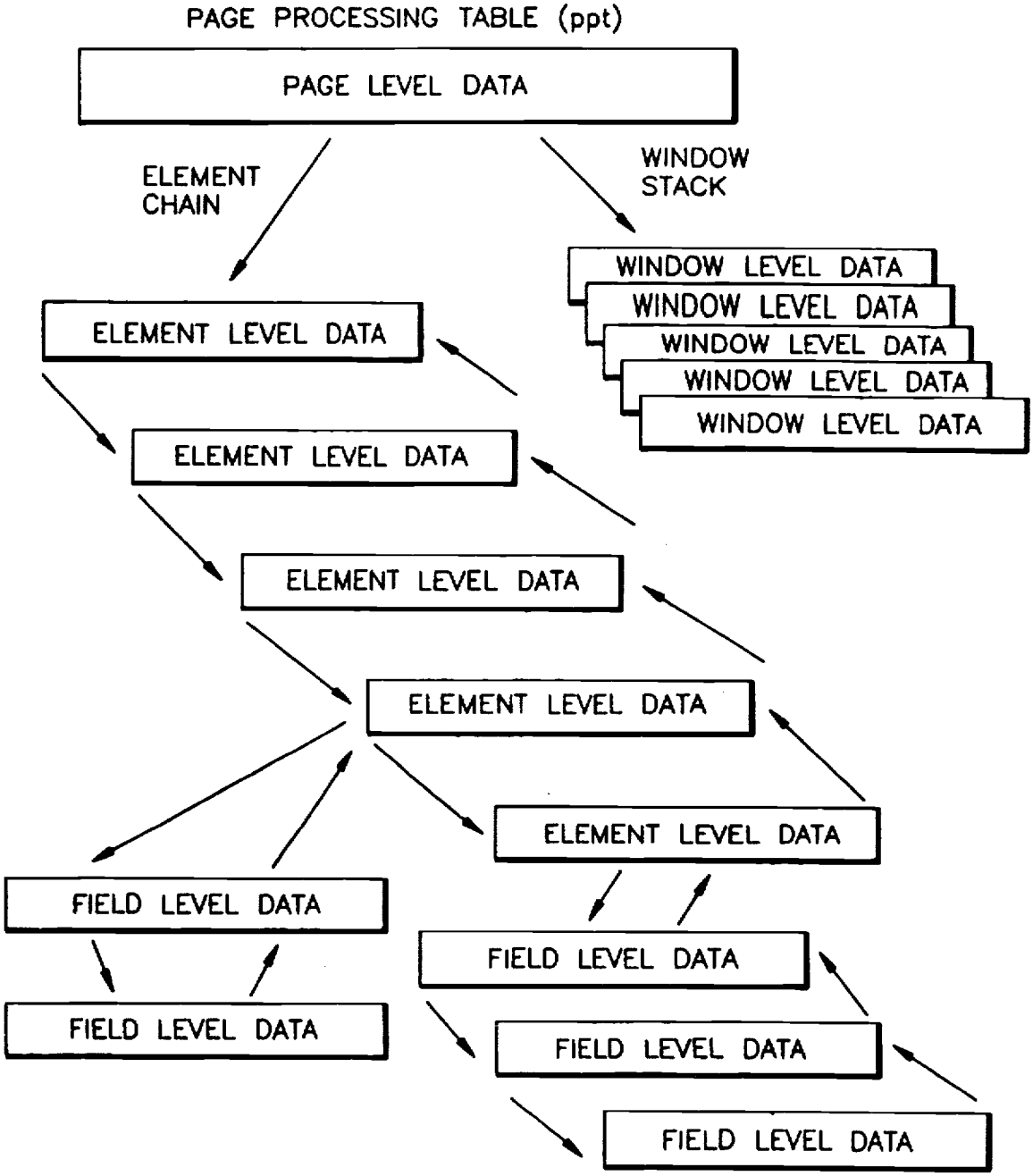
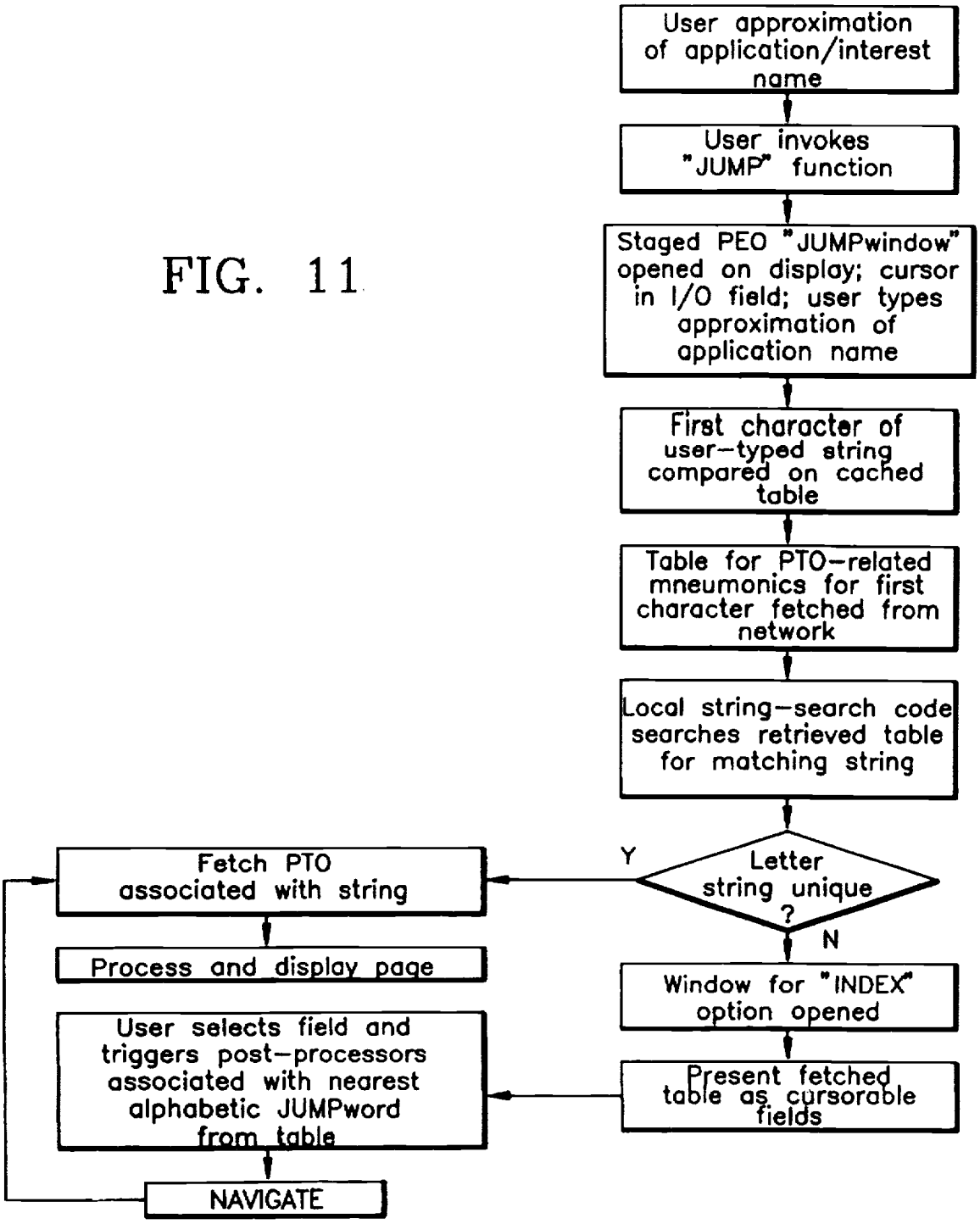

PAGE PROCESSING TABLE (ppt)
PAGE LEVEL DATA
ELEMENT CHAIN
WINDOW STACK
WINDOW LEVEL DATA
WINDOW LEVEL DATA
WINDOW LEVEL DATA
WINDOW LEVEL DATA
WINDOW LEVEL DATA
ELEMENT LEVEL DATA
ELEMENT LEVEL DATA
ELEMENT LEVEL DATA
ELEMENT LEVEL DATA
ELEMENT LEVEL DATA
FIELD LEVEL DATA
FIELD LEVEL DATA
FIELD LEVEL DATA
FIELD LEVEL DATA
FIELD LEVEL DATA

FIG. 11


User approximation of application/interest name
User invokes "JUMP" function
Staged PEO "JUMPwindow" opened on display; cursor in I/O field; user types approximation of application name
First character of user-typed string compared on cached table
Table for PTO-related mneumonics for first character fetched from network
Local string-search code searches retrieved table for matching string
Letter string unique ?
Y
N
Fetch PTO associated with string
Process and display page
User selects field and triggers post-processors associated with nearest alphabetic JUMPword from table
NAVIGATE
Window for "INDEX" option opened
Present fetched table as cursorable fields

METHOD FOR PRESENTING APPLICATIONS IN AN INTERACTIVE SERVICE

RELATED APPLICATIONS

This is a division of application Ser. No. 388,156 filed Jul. 28, 1989, which issued Sep. 13, 1994, as U.S. Pat. No. 5,347,632, application Ser. No. 388,156 being a continuation in part of application Ser. No. 328,790, filed Mar. 23, 1989, abandoned, which itself was a continuation in part of application Ser. No. 219,931, filed Jul. 15, 1988 abandoned.

BACKGROUND OF THE INVENTION

1.Field of Use

This invention relates generally to a method for presenting applications in a distributed processing, interactive computer network intended to provide very large numbers of simultaneous users; e.g. millions, access to an interactive service having large numbers; e.g., thousands, of applications which include pre-created, interactive text/graphic sessions; and more particularly, to a method for presenting applications, the method featuring step for generating a screen display at respective user reception systems, the screen display including a plurality of partitions for concurrently presenting at least a user-requested application and a group of command functions for managing the display, the group of command functions including a subgroup of functions for randomly selecting applications for display with a variety of different procedures, the method also including steps for opening and closing windows on the display for presenting data relating to the displayed applications, and further partitions for concurrently displaying, for example, an additional application which may include advertising.

2. Prior Art

Interactive computer networks are not new. Traditionally they have included conventional, hierarchical architectures wherein a central, host computer responds to the information requests of multiple users. An illustration would be a time-sharing network in which multiple users, each at a remote terminal, log onto a host that provides data and software resource for sequentially receiving user data processing requests, executing them and supplying responses back to the users.

While such networks have been successful in making the processing power of large computers available to many users, problems have existed with them. For example, in such networks, the host has been required to satisfy all the user data processing requests. As a result, processing bottlenecks arise at the host that cause network slowdowns and compel expansion in computing resources; i.e., bigger and more complex computer facilities, where response times are sought to be held low in the face of increasing user populations.

Host size and complexity, however, are liabilities for interactive networks recently introduced to offer large numbers of the public access to transactional services such as home shopping, banking, and investment maintenance, as well as informational services concerning entertainment, business and personal matters. As can be appreciated, commercial interactive networks will have to provide attractive services at low cost and with minimal response times in order to be successful. Unlike military and governmental networks where, because of the compulsory nature of the service performed, costs, content and efficiency are of secondary concern, in commercial services, since use is predominantly elective, and paid for by the consumer, costs will have to be held low, content made interesting and response times reduced in order to attract and hold both users who would subscribe to the service and merchandisers who would rely on it as a channel of distribution for their good and services.

In this regard, it has been found particularly important that the service interface; i.e., screen presentation and manipulation facility, enable the user to quickly and easily understand and control what is displayed. Since an interactive services represent an array of informational and transactional services intended to be embraced in a matrix of entertainment, it is essential that the user find the service simple and enjoyable to work with. If the user finds the service awkward in expression or difficult to manipulate, interest and participation soon wane, rendering it impossible to maintain the broad base subscriber and merchandiser support necessary to sustain viability.

However, providing clarity of presentation and ease of use along with comprehensive services and economy of expression, a combination essential to success, is difficult. In the past service offerors have relied on full-screen, continuous-stream displays that scroll past the user and require user recall for integration of multi-screen matter. In addition, the command structure for controlling such services has tended to be either ponderous, discrete, full-screen menus that interrupt subject matter presentation and stream of impression, or cryptic, key-combination commands that rely on user memory for application. Accordingly, interacting with such services can be more work than play, tending, at best to blunt enthusiasm, or at worst discourage it. Still further, the over reliance on user memory tends not only to color the experience as laborious, but also to both slowdown presentation as users attempt to fully understand the matter on the first pass, or require the user review the material in several passes so that a clear impression can be taken, both of which drive up the service usage costs rendering the service commercially unattractive.

SUMMARY OF INVENTION

Accordingly, it is an object of this invention to provide a method for presenting applications in an interactive service which enables clarity of application expression.

It is another object of this invention to provide a method for presenting applications in an interactive service which enables ease of service manipulation.

It is still another object of this invention to provide a method for presenting applications in an interactive service that places reduced reliance on user memory for application comprehension and exercise of service control commands.

It is yet another object of this invention to provide a method for presenting applications in an interactive service which enables concurrent, on-screen display of applications and service control commands.

It is a further object of this invention to provide a method for presenting applications in an interactive service which enables forward and reverse movement between multiple application display screens.

It is a still further object of this invention to provide a method for presenting applications in an interactive service which permits multiple procedures for navigating to successive applications.

And, it is still further object of this invention to provide a method for presenting applications in an interactive service that enables concurrent display of multiple applications, at least one of which may include advertising.

Briefly, the method for presenting applications in an interactive service in accordance with this invention achieves the above-noted and other objects by featuring steps for generating a screen displays at respective user reception systems, the screen display including a plurality of partitions for concurrently presenting at least a user-requested application and a group of command functions for managing the display. In accordance with the invention, the command functions include a subgroup of functions that facilitate random navigation to new applications, at the user's behest, employing a variety of different procedures. As well, the method includes steps for opening and closing windows on the display to enable presentation of additional data relating to the presented applications. Still further, the method includes steps for providing additional partitions for concurrently displaying other applications, which may include advertising.

In preferred form, the method features steps for presenting the command function in a command bar fixed-located on the display screen; e.g., at the screen bottom, concurrently with the displayed application so that the user can conduct display control functions while viewing an application. Additionally, in preferred form, the command bar features functions for progressing forward and backward in the application, thus reducing the need for user reliance on memory or repeat of the entire application to aid comprehension.

Also in preferred form, the method features steps for presenting a subgroup of commands at the command bar for enabling the user to randomly navigate to other available applications. Particularly, the navigation subgroup includes a command entitled "Path" which enables the user to sequence through a list of user-designated preferred applications, that, in effect, "program", progress through the service, thus enabling the user to easily and quickly review the service, as for example, on a daily basis, for news, financial, sports, business and other information.

Still further, the method features a navigation subgroup command entitled "Jump" which opens a window at the display concurrent with the application, which enables the user to select a new application for display based on a review of the available applications using either a string-descriptor search, an alphabetical search, a subject category search or a physical analogy; e.g., application to departments of a store, etc. Also in preferred form, the method features steps for opening windows over the currently displayed application to present further information concerning the application or facilitate the undertaking of interactive operation with respect to the application. As will be appreciated, this again reduces need for user reliance on memory, as it enables the providing of a reference that keeps the user oriented in progressing through a service session.

Additionally, the method of the present invention enables yet additional application to be concurrently presented at the display screen; as for example applications concerning advertising which may be of interest to the user. In these advertising-related application, the user can either obtain additional information or, if desired, undertake transactional event; as for example, buying goods or services.

DESCRIPTION OF THE DRAWINGS

The above and further objects, features and advantages of the invention will become clear from the following more detailed description when read with reference to the accompanying drawings in which:

FIG. **1** is a block diagram of the interactive computer network in which the application-presentation method of the present invention may be employed;

FIG. **2** is a schematic diagram of the network illustrated in FIG. **1**;

FIGS. **3***a* and **3***b* are plan views of a display screen for a user reception system employed in a network in which the application-presentation method of the present invention may be practiced;

FIGS. **4***a*, **4***b*, **4***c* and **4***d* are schematic drawings that illustrate the structure of objects, and object segments that may be used in a network in which the application-presentation method of the present invention may be employed;

FIG. **5***a* is a schematic diagram that illustrates the configuration of the page template object which might be used for presentation of an application in a network in which the application-presentation method of the present invention may be practiced;

FIG. **5***b* is a schematic diagram that illustrates page composition which might be used for presentation of applications in a network in which the application-presentation method of the present invention may be practiced;

FIG. **6** is a schematic diagram that illustrates the protocol which might be used by a reception system for supporting applications in a network in which the application-presentation method of the present invention may be practiced;

FIG. **7** is a schematic diagram that illustrates major layers for a reception system which might be used for supporting applications in a network in which the application-presentation method of the present invention may be practiced;

FIG. **8** is a block diagram that illustrates native code modules for a reception system which might be used for supporting applications in a network in which the application-presentation method of the present invention may be practiced;

FIG. **9** is a schematic diagram that illustrates an example of a partitioned application to be processed by a reception system which might be used for supporting applications in a network in which the application-presentation method of the present invention may be practiced;

FIG. **10** illustrates generation of a page with a page processing table for a reception system which might be used for supporting applications in a network in which the application-presentation method of the present invention may be practiced;

FIG. **11** is a flow diagram for an aspect of the navigation method of a reception system which might be used for supporting applications in a network in which the application-presentation method of the present invention may be practiced.

DESCRIPTION OF THE PREFERRED EMBODIMENT

GENERAL SYSTEM DESCRIPTION

FIGS. **1** and **2** show a network in which the method of the current invention for presenting applications might be used. As seen the network, designated **10**, includes a plurality of reception units within a reception layer **401** for displaying information and providing transactional services. In this arrangement, many users each access network **10** with a conventional personal computer; e.g., one of the IBM or IBM-compatible type, which has been provided with application software to constitute a reception system (RS) **400**.

As seen in FIG. **1**, interactive network **10** uses a layered structure that includes an information layer **100**, a switch/file

server layer **200**, and cache/concentrator layer **300** as well as reception layer **401**. This structure maintains active application databases and delivers requested parts of the databases on demand to the plurality of RSs **400**, shown in FIG. **2**. As seen in FIG. **2**, cache/concentrator layer **300** includes a plurality of cache/concentrator units **302**, each of which serve a plurality of RS **400** units over lines **301**. Additionally, switch/file server layer **200** is seen to include a server unit **205** connected to multiple cache/concentrator units **302** over lines **201**. Still further, server unit **205** is seen to be connected to information layer **100** and its various elements, which act as means for producing, supplying and maintaining the network databases and other information necessary to support network **10**. Continuing, switch/filer layer **200** is also seen to include gateway systems **210** connected to server **205**. Gateways **210** couple layer **200** to other sources of information and data; e.g., other computer systems. As will be appreciated by those skilled in the art, layer **200**, like layers **401** and **300**, could also include multiple servers, gateways and information layers in the event even larger numbers of users were sought to be served.

Continuing with reference to FIG. **2**, in preferred form, each RS **400** is seen to include a personal computer **405** having a CPU **410** including a microprocessor (as for example, one of the types made by INTEL Corporation in its X'86 family of microprocessors), companion RAM and ROM memory and other associated elements, such as monitor **412** with screen **414** and a keyboard **424**. Further, personal computer **405** may also include one or two floppy disk drives **416** for receiving diskettes **426** containing application software used to support the interactive service and facilitate the interactive sessions with network **10**. Additionally, personal computer **405** would include operating systems software; e.g., MS-DOS, supplied on diskettes **428** suitable for the personal computer being used. Personal computer **405** still further may also include a hard-disk drive **420** for storing the application software and operating system software which may be transferred from diskettes **426** and **428** respectfully.

Once so configured, each RS **400** provides: a common interface to other elements of interactive computer network **10**; a common environment for application processing; and a common protocol for user-application conversation which is independent of the personal computer brand used. RS **400** thus constitutes a universal terminal for which only one version of all applications on network **10** need be prepared, thereby rendering the applications interpretable by a variety of brands of personal computers.

RS **400** formulated in this fashion is capable of communication with the host system to receive information containing either of two types of data, namely objects and messages. Objects have a uniform, self-defining format known to RS **400**, and include data types, such as interpretable programs and presentation data for display at monitor screen **414** of the user's personal computer **405**. Applications presented at RS **400** are partitioned into objects which represent the minimal units available from the higher levels of interactive network **10** or RS **400**. In this arrangement, each application partition typically represents one screen or a partial screen of information, including fields filled with data used in transactions with network **10**. Each such screen, commonly called a page, is represented by its parts and is described in a page template object, discussed below.

Applications, having been partitioned into minimal units, are available from higher elements of network **10** or RS **400**, and are retrieved on demand by RS **400** for interpretive execution. Thus, not all partitions of a partitioned application need be resident at RS **400** to process a selected partition, thereby raising the storage efficiency of the user's RS **400** and minimizing response time. Each application partition is an independent, self-contained unit and can operate correctly by itself. Each partition may refer to other partitions either statically or dynamically. Static references are built into the partitioned application, while dynamic references are created from the execution of program logic using a set of parameters, such as user demographics or locale. Partitions may be chosen as part of the RS processing in response to user created events, or by selecting a key word of the partitioned application (e.g., "JUMP" or "INDEX," discussed below), which provides random access to all services represented by partitioned applications having key words.

Objects provide a means of packaging and distributing partitioned applications. As noted, objects make up one or more partitioned applications, and are retrieved on demand by a user's RS **400** for interpretive execution and selective storage. All objects are interpreted by RS **400**, thereby enabling applications to be developed independently of the personal computer brand used.

Objects may be nested within one another or referenced by an object identifier (object-id) from within their data structure. References to objects permit the size of objects to be minimized. Further, the time required to display a page is minimized when referenced objects are stored locally at RS **400** (which storage is determined by prior usage meeting certain retention criteria to be described more fully below), or have been pre-fetched, or in fact, are already used for the current page.

Objects carry application program instructions and/or information for display at monitor screen **414** of RS **400**. Application program objects, called pre-processors and post-processors, set up the environment for the user's interaction with network **10** and respond to events created when the user inputs information at keyboard **424** of RS **400**. Such events typically trigger a program object to be processed, causing one of the following: sending of transactional information to the coapplications in one layer of the network **10**; the receiving of information for use in programs or for presentation in application-dependent fields on monitor screen **414**; or the requesting of a new objects to be processed by RS **400**. Such objects may be part of the same application or a completely new application.

The RS **400** supports a protocol by which the user and the partitioned applications communicate. All partitioned applications are designed knowing that this protocol will be supported in RS **400**. Hence, replication of the protocol in each partitioned application is avoided, thereby minimizing the size of the partitioned application.

RS **400** includes a means to communicate with network **10** to retrieve objects in response to events occurring at RS **400** and to send and receive messages.

RS **400** includes a means to selectively store objects according to a predetermined storage criterion, thus enabling frequently used objects to be stored locally at the RS, and causing infrequently used objects to forfeit their local storage location. The currency of objects stored locally at the RS **400** is verified before use according to the object's storage control parameters and the storage criterion in use for version checking.

Selective storage tailors the contents of the RS **400** memory to contain objects representing all or significant parts of partitioned applications favored by the user. Because selective storage of objects is local, response time

is reduced for those partitioned applications that the user accesses most frequently.

Since much of the application processing formerly done by a host computer in previously known time-sharing networks is now performed at the user's RS **400**, the higher elements of network **10**, particularly layer **200**, have as their primary functions the routing of messages, serving of objects, and line concentration. The narrowed functional load of the higher network elements permits many more users to be serviced within the same bounds of computer power and I/O capability of conventional host-centered architectures.

Network **10** provides information on a wide variety of topics, including, but not limited to news, industry, financial needs, hobbies and cultural interests. Network **10** thus eliminates the need to consult multiple information sources, giving users an efficient and timesaving overview of subjects that interest them.

The transactional features of interactive network **10** saves the user time, money, and frustration by reducing time spent traveling, standing in line, and communicating with sales personnel. The user may, through RS **400**, bank, send and receive messages, review advertising, place orders for merchandise, and perform other transactions.

In preferred form, network **10** provides information, advertising and transaction processing services for a large number of users simultaneously accessing the network via the public switched telephone network (PSTN), broadcast, and/or other media with their RS **400** units. Services available to the user include display of information such as movie reviews, the latest news, airlines reservations, the purchase of items such as retail merchandise and groceries, and quotes and buy/sell orders for stocks and bonds. Network **10** provides an environment in which a user, via RS **400** establishes a session with the network and accesses a large number of services. These services are specifically constructed applications which as noted are partitioned so they may be distributed without undue transmission time, and may be processed and selectively stored on a user's RS **400** unit.

SYSTEM CONFIGURATION

As shown in FIG. **1**, interactive computer network **10** includes four layers: information layer **100**, switch and file server layer **200**, concentrator layer **300**, and reception layer **401**.

Information layer **100** handles: (1) the production, storage and dissemination of data and (2) the collection and off-line processing of such data from each RS session with the network **10** so as to permit the targeting of information and advertising to be presented to users and for traditional business support.

Switch and file server layer **200** and cache/concentrator layer **300** together constitute a delivery system **20** which delivers requested data to the RSs **400** of reception layer **401** and routes data entered by the user or collected at RSs **400** to the proper application in network **10**. With reference to FIG. **2**, the information used in a RS **400** either resides locally at the RS **400**, or is available on demand from the cache/concentrator **300** or the file server **205**, via the gateway **210**, which may be coupled to external providers, or is available from information layer **100**.

There are two types of information in the network **10** which are utilized by the RS **400**: objects and messages.

Objects include the information requested and utilized by the RS **400** to permit a user to select specific parts of applications, control the flow of information relating to the applications, and to supply information to the network. Objects are self-describing structures organized in accordance with a specific data object architecture, described below. Objects are used to package presentation data and program instructions required to support the partitioned applications and advertising presented at a RS **400**. Objects are distributed on demand throughout interactive network **10**. Objects may contain: control information; program instructions to set up an application processing environment and to process user or network created events; information about what is to be displayed and how it is to be displayed; references to programs to be interpretively executed; and references to other objects, which may be called based upon certain conditions or the occurrence of certain events at the user's personal computer, resulting in the selection and retrieval of other partitioned applications packaged as objects.

Messages are information provided by the user or the network and are used in fields defined within the constructs of an object, and are seen on the user's RS monitor **412**, or are used for data processing at RS **400**. Additionally, and as more fully described hereafter, messages are the primary means for communication within and without the network. The format of messages is application dependent. If the message is input by the user, it is formatted by the partitioned application currently being processed on RS **400**. Likewise, and with reference to FIG. **2**, if the data are provided from a co-application database residing in delivery system **20**, or accessed via gateway **210** or high function system **110** within the information layer **100**, the partitioned application currently being processed on RS **400** causes the message data to be displayed in fields on the user's display monitor as defined by the particular partitioned application.

All active objects reside in file server **205**. Inactive objects or objects in preparation reside in producer system **120**. Objects recently introduced into delivery system **20** from the producer system **120** will be available from file server **205**, but, may not be available on cache/concentrator **302** to which the user's RS **400** has dialed. If such objects are requested by the RS **400**, the cache/concentrator **302** automatically requests the object from file server **205**. The requested object is routed back to the requesting cache/concentrator **302**, which automatically routes it to the communications line on which the request was originally made, from which it is received by the RS **400**.

The RS **400** is the point of application session control because it has the ability to select and randomly access objects representing all or part of partitioned applications and their data. RS **400** processes objects according to information contained therein and events created by the user on personal computer **405**.

Applications on network **10** act in concert with the distributed partitioned applications running on RS **400**. Partitioned applications constructed as groups of objects and are distributed on demand to a user's RS **400**. An application partition represents the minimum amount of information and program logic needed to present a page or window, i.e. portion of a page presented to the user, perform transactions with the interactive network **10**, and perform traditional data processing operations, as required, including selecting another partitioned application to be processed upon a user generated completion event for the current partitioned application.

Objects representing all or part of partitioned applications may be stored in a user's RS **400** if the objects meet certain

criteria, such as being non-volatile, non-critical to network integrity, or if they are critical to ensuring reasonable response time. Such objects are either provided on diskettes **426** together with RS **400** system software used during the installation procedure or they are automatically requested by RS **400** when the user makes selections requiring objects not present in RS **400**. In the latter case, RS **400** requests from cache/concentrator layer **300** only the objects necessary to execute the desired partitioned application.

Reception system application software **426** in preferred form is provided for IBM and IBM-compatible brands of personal computers **405**, and all partitioned applications are constructed according to a single architecture which each such RS **400** supports. With reference to FIG. **2**, to access network **10**, a user preferably has a personal computer **405** with at least 512K RAM and a single disk drive **416**. The user typically accesses network **10** using a 1,200 or 2,400 bps modem (not shown). To initiate a session with network **10**, objects representing the logon application are retrieved from the user's personal diskette, including the R.S. application software, which was previously set up during standard installation and enrollment procedures with network **10**. Once communication between RS **400** and cache/concentrator layer **300** has been established, the user begins a standard logon procedure by inputting a personal entry code. Once the logon procedure is complete, the user can begin to access various desired services (i.e., partitioned applications) which provide display of requested information and/or transaction operations.

APPLICATIONS AND PAGES

Applications, i.e., information events, are composed of a sequence of one or more pages opened at screen **414** of monitor **412**. This is better seen with reference to FIGS. **3***a* and **3***b* were a page **255** is illustrated as might appear at screen **414** of monitor **412**. With reference to FIG. **3***a*, in accordance with the invention, each page **255** is formatted with a service interface having page partitions **250**, **260**, **280**, and **290** (not to be confused with application partitions). Window page partitions **275**, well known in the art, are also available and are opened and closed conditionally on page **255** upon the occurrence of an event specified in the application being run. Each page partition **250**, **260**, **280**, and **290** and window **275** is made up of a page element which defines the content of the partition or window.

In preferred form, each page **255** includes: a header page partition **250**, which has a page element associated with it and which typically conveys information on the page's topic or sponsor; one or more body page partitions **260** and window page partitions **275**, each of which is associated with a page element which as noted gives the informational and transactional content of the page. For example, a page element may contain presentation data selected as a menu option in the previous page, and/or may contain prompts to which a user responds in pre-defined fields to execute transactions. As illustrated in FIG. **3***b*, the page element associated with body page partition **260** includes display fields **270**, **271**, **272**. A window page partition **275** seen in FIG. **3***a* represents the same informational and transactional capability as a body partition, except greater flexibility is provided for its location and size.

Continuing with reference to FIG. **3***a*, in accordance with the invention, advertising **280** is provided over network **10**, and, like page elements, also include information for display on page **255**, and may be included in any partition of a page. Advertising **280** is presented to the user on an individualized basis from queues of advertising object identifications (ids) that are constructed off-line by business system **130**, and sent to file server **205** where they are accessible to each RS **400**.

Individualized queues of advertising object ids are constructed based upon data collected on the partitioned applications that were accessed by a user, and upon events the user generated in response to applications. The data are collected and reported by RS **400** to a data collection co-application in file server **205** for later transmission to business system **130**. In addition to application access and use characteristics, a variety of other parameters, such as user demographics or postal ZIP code, may be used as targeting criteria. From such data, queues of advertising object ids are constructed that are targeted to either individual users or to sets of users who fall into certain groups according to such parameters. Stated otherwise, the advertising presented is individualized to the respective users based on characterizations of the respective users as defined by the interaction history with the service and such other information as user demographics and locale. As will be appreciated by those skilled in the art, conventional marketing analysis techniques can be employed to establish the user characterizations based on the collected application usage data above noted and other information.

Also with reference to FIG. **3***b*, the service interface is seen to include a command region **285** which enables the user to interact with the network RS **400** and other elements of network **10**, so as to cause such operations as navigating from page to page, performing a transaction, or obtaining more information about other applications. As shown in FIG. **3***b*, interface region **285** includes a command bar **290** having a number of commands **291–298** which the user can execute. The functions of commands **291–298** are discussed in greater detail below.

NETWORK OBJECTS

As noted above, in conventional time-sharing computer networks, the data and program instructions necessary to support user sessions are maintained at a central host computer. However, that approach has been found to create processing bottlenecks as greater numbers of users are connected to the network; bottlenecks which require increases in processing power and complexity; e.g., multiple hosts of greater computing capability, if the network is to meet demand. Further, such bottlenecks have been found to also slow response time as more users are connected to the network and seek to have their requests for data processing answered.

The consequences of the host processing bottlenecking is to either compel capital expenditures to expand host processing capability, or accept longer response times; i.e., a slower network, and risk user dissatisfaction.

However, even in the case where additional computing power is added, and where response time is allowed to increase, eventually the host becomes user saturated as more and more users are sought to be served by the network. The network described above, however, is designed to alleviate the effects of host-centered limitations, and extend the network saturation point. This objective is achieved by reducing the demand on the host for processing resources by structuring the network so that the higher network levels act primarily to maintain and supply data and programs to the lower levels of the network, particularly RS **400**, which acts to manage and sustain the user screen displays.

More particularly, the described network features procedures for parsing the network data and program instructions

required to support the interactive user sessions into packets, referred to as objects, and distributing them into the network where they can be processed at lower levels, particularly, reception system **400**.

The screens presented at the user's monitor are each divided into addressable partitions shown in FIG. **3***a*, and the display text and graphics necessary to make up the partitions, as well as the program instructions and control data necessary to deliver and sustain the screens and partitions are formulated from pre-created objects. Further, the objects are structured in accordance with an architecture that permits the displayed data to be relocatable on the screen, and to be reusable to make up other screens and other sessions, either as pre-created and stored sessions or interactive sessions, dynamically created in response to the user's requests.

As shown in FIG. **4***c*, the network objects are organized as a family of objects each of which perform a specific function in support of the interactive session. More particularly, in accordance with the preferred form, the network object family is seen to include 6 members: page format objects **502**, page element objects **504**, window objects **506**, program objects **508**, advertisement objects **510** and page template objects **500**.

Within this family, page format objects **502** are designed to define the partitioning **250** to **290** of the monitor screen shown in FIG. **3***a*. The page format objects **502** provide a means for pre-defining screen partitions and for ensuring a uniform look to the page presented on the reception system monitor. They provide the origin; i.e., drawing points, and dimensions of each page partition and different values for presentation commands such as palette and background color.

Page format objects **502** are referenced whenever non-window data is to be displayed and as noted ensure a consistent presentation of the page. In addition, page format objects **502** assures proper tessellation or "tiling" of the displayed partitions.

Page element objects **504**, on the other hand, are structured to contain the display data; i.e., text and graphic, to be displayed which is mapped within screen partitions **250** to **290**, and to further provide the associated control data and programs. More specifically, the display data is described within the object as NAPLPS data, and includes, PDI, ASCII, Incremental Point and other display encoding schemes. Page element objects also control the functionality within the screen partition by means of field definition segments **516** and program call segments **532**, as further described in connection with the description of such segments hereafter. Page element objects **504** are relocatable and may be reused by many pages. To enable the displayable data to be relocated, display data must be created by producers in the NAPLPS relative mode.

Continuing with reference to FIG. **4***c*, window objects **506** include the display and control data necessary to support window partitions **275** best seen in FIG. **3***a*. Windows contain display data which overlay the base page and control data which supersede the base page control data for the underlying screen during the duration of the window. Window objects **506** contain data which is to be displayed or otherwise presented to the viewer which is relatively independent from the rest of the page. Display data within windows overlay the base page until the window is closed. Logic associated with the window supersedes base page logic for the duration of the window. When a window is opened, the bit map of the area covered by window is saved and most logic functions for the overlaid page are deactivated. When the window is closed, the saved bit map is swapped onto the screen, the logic functions associated with the window are disabled, and prior logic functions are reactivated.

Windows are opened by user or program control. They do not form part of the base page. Windows would typically be opened as a result of the completion of events specified in program call segments **532**.

Window objects **506** are very similar in structure to page element objects **504**. The critical difference is that window objects **506** specify their own size and absolute screen location by means of a partition definition segment **528**.

Program objects **508** contain program instructions written in a high-level language called TRINTEX Basic Object Language, i.e., TBOL, described in greater detail hereafter, which may be executed on RS **400** to support the application. More particularly, program objects **508** include interpretable program code, executable machine code and parameters to be acted upon in conjunction with the presentation of text and graphics to the reception system monitors.

Program objects **508** may be called for execution by means of program call segments **532**, which specify when a program is to be executed (event), what program to execute (program pointer), and how programs should run (parameters).

Programs are treated as objects to conform to the open-ended design philosophy of the data object architecture (DOA), allowing the dissemination of newly developed programs to be easily and economically performed. As noted above, it is desirable to have as many of these program objects staged for execution at or as close to RS **400** as possible.

Still further, advertising objects **510** include the text and graphics that may be presented at ad partition **280** presented on the monitor screen as shown in FIG. **3***b*.

Finally, the object family includes page template objects **500**. Page template objects **500** are designed to define the components of the full screen presented to the viewer. Particularly, page template objects **500** include the entry point to a screen, the name of the page format objects which specify the various partitions a screen will have and the page element object that contain the display data and partitioning parameters for the page.

Additionally, page template object **500** includes the specific program calls required to execute the screens associated with the application being presented to the user, and may serve as the means for the user to selectively move through; i.e., navigate the pages of interest which are associated with various applications. Thus, in effect, page template objects **500** constitute the "recipe" for making up the collection of text and graphic information required to make the screens to be presented to the user.

Objects **500** to **510** shown in FIG. **4***c* are themselves made up of further sub-blocks of information that may be selectively collected to define the objects and resulting pages that ultimately constitute the application presented to the user in an interactive text and graphic session.

More specifically and as shown schematically in FIG. **4***a*, objects **500** to **510** are predefined, variable length records consisting of a fixed length header **551** and one or more self-defining record segments **552** a list of which is presented in FIG. **4***c* as segment types **512** to **540**.

In accordance with this design, and as shown in FIG. **4***b*, object header **551** in preferred form is 18 bytes in length and

contains a prescribed sequence of information which provides data regarding the object's identification, its anticipated use, association to other objects, its length and its version and currency.

More particularly, each of the 18 bytes of object header **551** are conventional hexadecimal, 8 bit bytes and are arranged in a fixed pattern to facilitate interpretation by network **10**. Particularly, and as shown in FIG. **4***b*, the first byte of header **551**; i.e., byte **1**, identifies the length of the object ID in hexadecimal. The next six bytes; i.e., bytes **2** to **7**, are allocated for identifying access control to the object so as to allow creation of closed user groups to whom the object(s) is to be provided. As will be appreciated by those skilled in the art, the ability to earmark objects in anticipation of user requests enables the network to anticipate requests and pre-collect objects from large numbers of them maintained to render the network more efficient and reduce response time. The following 4 bytes of header **551**; bytes **8** to **11**, are used to identify the set of objects to which the subject object belongs. In this regard, it will be appreciated that, again, for speed of access and efficiency of selection, the objects are arranged in groups or sets which are likely to be presented to user sequentially in presenting the page sets; i.e., screens that go to make up a session.

Following identification of the object set, the next byte in header **551**; i.e., byte **12**, gives the location of the subject object in the set. As will be appreciated here also the identification is provided to facilitate ease of object location and access among the many thousands of objects that are maintained to, thereby, render their selection and presentation more efficient and speedy.

Thereafter, the following byte of header **551**; i.e., byte **13**, designates the object type; e.g., page format, page template, page element, etc. Following identification of the object type, two bytes; i.e., bytes **14**, **15**, are allocated to define the length of the object, which may be of whatever length is necessary to supply the data necessary, and thereby provides great flexibility for creation of the screens. Thereafter, in accordance with the preferred form, a single byte; i.e., byte **16**, is allocated to identify the storage characteristic for the object; i.e., the criterion which establishes at what level in network **10** the object will be stored, and the basis upon which it will be updated. At least a portion of this byte; i.e, the higher order nibble (first 4 bits reading from left to right) is associated with the last byte; i.e., byte **18**, in the header which identifies the version of the object, a control used in determining how often in a predetermined period of time the object will be updated by the network.

Following storage characteristic byte **16**, header **551** includes a byte; i.e., **17**, which identifies the number of objects in the set to which the subject object belongs. Finally, and as noted above, header **551** includes a byte; i.e., **18**, which identifies the version of the object. Particularly the object version is a number to establish the control for the update of the object that are resident at RS **400**.

As shown in FIG. **4***a*, and as noted above, in addition to header **551**, the object includes one more of the various segment types shown in FIG. **4***c*.

Segments **512** to **540** are the basic building blocks of the objects. And, as in the case of the object, the segments are also self-defining. As will be appreciated by those skilled in the art, by making the segments self-defining, changes in the objects and their use in the network can be made without changing pre-existing objects.

As in the case of objects, the segments have also been provided with a specific structure. Particularly, and as shown in FIG. **4***a*, segments **552** consists of a designation of segment type **553**, identification of segment length **554**, followed by the information necessary to implement the segment and its associated object **555**; e.g., either, control data, display data or program code.

In this structure, segment type **553** is identified with a one-byte hexadecimal code which describes the general function of the segment. Thereafter, segment length **554** is identified as a fixed two-byte long field which carries the segment length as a hexadecimal number in INTEL format; i.e., least significant byte first. Finally, data within segments may be identified either by position or keyword, depending on the specific requirements of the segment.

The specific structure for the objects and segments in shown in FIG. **4***c* and is described below. In that description the following notation convention is used:

```
<>          - mandatory item
( )         - optional item
...         - item may be repeated
|item|  |item|
<  >  (  )  - items in a column indicate either/or
|item|  |item|
```

The structure for objects is:

PAGE TEMPLATE OBJECT.

[<header> (compression descriptor) <page format call> (page element call) . . . (program call) . . . (page element selector) (system table call) . . . external reference) (keyword/navigation) . . .];

As noted above, page format objects **502** are designed to define the partitioning **250** to **290** of monitor screen **414** shown in FIG. **3***a*.

PAGE FORMAT OBJECT.

[<header> (compression descriptor) (page defaults) <partition definition>];

PAGE ELEMENT OBJECT.

[<header> (compression descriptor) (presentation data) . . . (program call) . . . (custom cursor) . . . (custom text) . . . (field definition) . . . (field-level program call) . . . (custom cursor type 2) . . . (custom graphic) . . . (field definition type 2) . . . (array definition) . . . (inventory control)];

Page element objects, as explained, are structured to contain the display data; i.e., text and graphics, to be presented at screen partitions **250** to **290**.

WINDOW OBJECT.

[<header> (compression description) <partition definition> (page element call) (presentation data) . . . (program call) . . . (custom cursor) . . . (custom text) . . . (custom cursor type 2) . . . (custom graphic) . . . (field definition) . . . (field level program call) . . . (field definition type 2) . . . (array definition) . . . (inventory control)];

As noted, window objects include display and control data necessary to support window partition at screen **414**.

PROGRAM OBJECTS.

|<header> (compression descriptor) <program data> . . .].

Program objects, on the other hand, contain program instructions written in higher-level language which may be executed at RS **400** to support the application.

ADVERTISEMENT OBJECT.

[<header> (compression descriptor) (presentation data) . . . (program call) . . . (custom cursor) . . . (custom text) . . . (field definition) . . . (field-level program call) . . . (custom cursor type 2) . . . (custom graphic) . . . (field definition type 2) . . . (array definition) . . . (inventory control)];

and as can be seen, advertisement objects are substantially the same as page element objects, with the difference being

that, as their name implies, their subject matter is selected to concern advertising.

Continuing, the structure for the object segments follows from the above description, and is as described more fully in parent application Ser. No. 388,156 now issued as U.S. Pat. No. 5,347,632the contents of which patent are incorporated herein by reference.

NETWORK MESSAGES

In addition to the network objects, and the display data, control data, and the program instructions they contain as previously described, network **10** also exchanges information regarding the support of user sessions and the maintenance of the network as "messenger". Specifically, messages typically relate to the exchange of information associated with initial logon of a reception system **400** to network **10**, dialogue between RS **400** and other elements and communications by the other network elements amongst themselves.

To facilitate message exchange internally, and through gateway **210** to entities externally to network **10**, a protocol termed the "Data Interchange Architecture" (DIA) is used to support the transport and interpretation of information. More particularly, DIA enables: communications between RS **400** units, separation of functions between network layers **100**, **200**, **300** and **401**; consistent parsing of data; an "open" architecture for network **10**; downward compatibility within the network; compatibility with standard industry protocols such as the IBM System Network Architecture; Open Systems Interconnections standard; support of network utility sessions; and standardization of common network and application return codes.

Thus DIA binds the various components of network **10** into a coherent entity by providing a common data stream for communications management purposes. DIA provides the ability to route messages between applications based in IBM System Network Architecture (SNA), (well known in the art, and more fully described in *Data and Computer Communications*, by W. Stallings, Chapter 12, McMillian Publishing, Inc. (1985)) and non-SNA reception system applications; e.g. home computer applications. Further, DIA provides common data structure between applications run at RS **400** units and applications that may be run on external computer networks; e.g. Dow Jones Services, accessed through gateway **210**. As well, DIA provides support for utility sessions between backbone applications run within network **10**. A more detailed description of network messaging in provided in parent application Ser. No. 388,156 now issued as U.S. Pat. No. 5,347,632, the contents of which patent are incorporated herein by reference.

OBJECT LANGUAGE

In accordance with the design of network **10**, in order to enable the manipulation of the network objects, the application programs necessary to support the interactive text/graphic sessions are written in a high-level language referred to as "TBOL", (TRINTEX Basic Object Language, "TRINTEX" being the former company name of one of the assignees of this invention). TBOL is specifically adapted for writing the application programs so that the programs may be compiled into a compact data stream that can be interpreted by the application software operating in the user personal computer, the application software being designed to establish the network Reception System **400** previously noted and described in more detail hereafter.

The Reception System application software supports an interactive text/graphics sessions by managing objects. As explained above, objects specify the format and provide the content; i.e., the text and graphics, displayed on the user's screen so as to make up the pages that constitute the application. As also explained, pages are divided into separate areas called "partitions" by certain objects, while certain other objects describe windows which can be opened on the pages. Further, still other objects contain TBOL application programs which facilitate the data processing necessary to present the pages and their associated text and graphics.

As noted, the object architecture allows logical events to be specified in the object definitions. An example of a logical event is the completion of data entry on a screen; i.e., an application page. Logical events are mapped to physical events such as the user pressing the <ENTER> key on the keyboard. Other logical events might be the initial display of a screen page or the completion of data entry in a field. Logical events specified in page and window object definitions can be associated with the call of TBOL program objects.

RS **400** is aware of the occurrence of all physical events during the interactive text/graphic sessions. When a physical event such as depression of the forward <TAB> key corresponds to a logical event such as completion of data entry in a field, the appropriate TBOL program is executed if specified in the object definition. Accordingly, the TBOL programs can be thought of as routines which are given control to perform initialization and post-processing application logic associated with the fields, partitions and screens at the text/graphic sessions.

Reception System **400** run time environment uses the TBOL programs and their high-level key-word commands called verbs to provide all the system services needed to support a text/graphic session, particularly, display management, user input, local and remote data access.

TBOL programs have a structure that includes three sections: a header section in which the program name is specified; a data section in which the data structure the program will use are defined; and a code section in which the program logic is provided composed of one or more procedures. More specifically, the code section procedures are composed of procedure statements, each of which begins with a TBOL key-word called a verb.

The name of a procedure can also be used as the verb in a procedure statement exactly as if it were a TBOL key-word verb. This feature enables a programmer to extend the language vocabulary to include customized application-oriented verb commands.

Continuing, TBOL programs have a program syntax that includes a series of "identifiers" which are the names and labels assigned to programs, procedures, and data structures.

An identifier may be up to 31 characters long; contain only uppercase or lowercase letters A through Z, digits 0 through 9, and/or the special character underscore (_); and must begin with a letter. Included among the system identifiers are: "header section identifiers" used in the header section for the program name; "data section identifiers" used in the data section for data structure names, field names and array names; and finally, "code section identifiers" used in the code section for identification of procedure names and statement labels. A more detailed description of TBOL is provided in parent application Ser. No. 388,156 now issued as U.S. Pat. No. 5,347,632, the contents of which patent are incorporated herein by reference.

RECEPTION SYSTEM OPERATION

RS **400** of computer system network **10** uses software called native code modules (described below) to enable the

user to select options and functions presented on the monitor of personal computer **405**, to execute partitioned applications and to process user created events, enabling the partitioned application to interact with network **10**. Through this interaction, the user is able to input data into fields provided as part of the display, or may individually select choices causing a standard or personalized page to be built (as explained below) for display on the monitor of personal computer **405**. Such inputs will cause RS **400** to interpret events and trigger pre-processors or post-processors, retrieve specified objects, communicate with system components, control user options, cause the display of advertisements on a page, open or close window partitions to provide additional navigation possibilities, and collect and report data about events, including certain types of objects processed. For example, the user may select a particular option, such as opening or closing window partition **275**, which is present on the monitor screen **414** and follow the selection with a completion key stroke, such as ENTER. When the completion keystroke is made, the selection is translated into a logical event that triggers the execution of a post-processor (i.e., a partitioned application program object) to process the contents of the field.

In accordance with the invention, functions supporting the user-partitioned application interface can be performed using the command bar **290**, or its equivalent using pull down windows or an overlapping cascade of windows. These functions can be implemented as part of the RS native functions or can be treated as another partition(s) defined for every page for which an appropriate set of supporting objects exist and remain resident at RS **400**. If the functions are part of RS **400**, they can be altered or extended by verbs defined in the RS virtual machine that permit the execution of program objects to be triggered when certain functions are called, providing maximum flexibility.

To explain the functions the use of a command bar is assumed. Command bar **290** is shown in FIGS. **3***a* and **3***b* and includes a NEXT command **291**, a BACK command **292**, a PATH command **293**, a MENU command **294**, an ACTION command **295**, a JUMP command **296**, a HELP command **297**, and an EXIT command **298**.

NEXT command **291** causes the next page in the current page set to be built. If the last page of a page set has already been reached, NEXT command **291** is disabled by RS **400**, avoiding the presentation of an invalid option.

BACK command **292** causes the previous page of the current page set to be built. If the present page is the first in the page set, BACK command **292** is disabled, since it is not a valid option.

A filter program can be attached to both the NEXT or BACK functions to modify their implicit sequential nature based upon the value of the occurrence in the object set id.

PATH command **293** causes the next page to be built and displayed from a list of pages that the user has entered, starting from the first entry for every new session.

MENU command **294** causes the page presenting the previous set of choices to be rebuilt.

ACTION command **295** initiates an application dependent operation such as causing a new application partition to be interpreted, a window partition **275** to be opened and enables the user to input any information required which may result in a transaction or selection of another window or page.

JUMP command **296** causes window partition **275** to be opened, allowing the user to input a keyword or to specify one from an index that may be selected for display.

HELP command **297** causes a new application partition to be interpreted such as a HELP window pertaining to where the cursor is positioned to be displayed in order to assist the user regarding the present page, a particular partition, or a field in a page element.

EXIT command **298** causes a LOGOFF page template object (PTO) to be built, and a page logoff sequence to be presented at RS **400** monitor screen **414**.

NAVIGATION INTERFACE

Continuing, as a further feature, network **10** includes an improved procedure for searching and retrieving applications from the store of applications distributed throughout network **10**; e.g., server **205**, cache/concentrator **302** and RS **400**. More specifically, the procedure features use of precreated search tables which represent subsets of the information on the network arranged with reference to the page template objects (PTO) and object-ids of the available applications so that in accordance with the procedure, the relevant tables and associated objects can be provided to and searched at the requesting RS **400** without need to search the entire store of applications on the network. As will be appreciated, this reduces the demand on the server **205** for locating and retrieving applications for display at monitor **412**.

In conventional time-sharing networks that support large conventional databases, the host receives user requests for data records; locates them; and transmits them back to the users. Accordingly, the host is obliged to undertake the data processing necessary to isolate and supply the requested information. And, as noted earlier, where large numbers of users are to be served, the many user requests can bottleneck at the host, taxing resources and leading to response slowdown.

Further, users have experienced difficulty in searching databases maintained on conventional time-sharing networks. For example, difficulties have resulted from the complex and varied way previously known database suppliers have organized and presented their information. Particularly, some database providers require searching be done only in selected fields of the database, thus requiring the user to be fully familiar with the record structure. Others have organized their databases on hierarchial structures which require the user understand the way the records are grouped. Still further, yet other database suppliers rely upon keyword indices to facilitate searching of their records, thus requiring the user to be knowledgeable regarding the particular keywords used by the database provider.

Network **10**, however, is designed to avoid such difficulties. In the preferred form, the network includes procedures for creating preliminary searches which represent subsets of the network applications users are believed likely to investigate. Particularly, in accordance with these procedures, for the active applications available on network **10**, a library of tables is prepared, and maintained within each of which a plurality of so called "keywords" are provided that are correlated with page template objects and object-ids of the entry screen (typically the first screen) for the respective application. In the preferred embodiment, approximately 1,000 tables are used, each having approximately 10 to 20 keywords arranged in alphabetical order to abstract the applications on the network. Further, the object-id for each table is associated with a code in the form of a character string mnemonic which is arranged in a set of alphabetically sequenced mnemonics termed the sequence set so that on entry of a character string at an RS **400**, the object-id for the

relevant keyword table can be obtained from the sequence set. Once the table object-id is identified, the keyword table corresponding to the desired subset of the objects and associated applications can then be obtained from network **10**. Subsequently the table can be presented to the user's RS **400**, where the RS **400** can provide the data processing required to present the potentially relevant keywords, objects and associated applications to the user for further review and determination as to whether more searching is required. As will be appreciated, this procedure reduces demand on server **205** and thereby permits it to be less complex and costly, and further, reduces the likelihood of host overtaxing that may cause network response slowdown.

As a further feature of this procedure, the library of keywords and their associated PTOs and objects may be generated by a plurality of operations which appear at the user's screen as different search techniques. This permits the user to select a search technique he is most comfortable with, thus expediting his inquiry.

More particularly, the user is allowed to invoke the procedure by calling up a variety of operations. The various operations have different names and seemingly present different search strategies. Specifically, the user may invoke the procedure by initiating a "Jump" command at RS **400**. Thereafter, in connection with the Jump operation, the user, when prompted, may enter a word of the user's choosing at monitor screen **414** relating to the matter he is interested in locating; i.e., a subject matter search of the network applications. Additionally, the users may invoke the procedure by alternatively calling up an operation termed "Index" with selection of the Index command. When selected, the Index command presents the user with an alphabetical listing of keywords from the tables noted above which the user can select from; i.e., an alphabetical search of the network applications. Further, the user may evoke the procedure by initiating an operation termed "Guide." By selecting the Guide command, the user is provided with a series of graphic displays that presents a physical description of the network applications; e.g., department floor plan for a store the user may be electronically shopping in. Still further, the user may invoke the procedures by initiating an operation termed "Directory." By selecting the Directory command, the user is presented with the applications available on the network as a series of hierarchial menus which present the content of the network information in commonly understood categories. Finally, the user may invoke the procedure by selecting the "Path" command, which accesses a list of keywords the user has previously selected; i.e., a personally tailored form of the Index command described above. As described hereafter, Path further includes a Viewpath operation which permits the user to visually access and manage the Path list of keywords. In preferred form, where the user has not selected a list of personalized keywords, a default set is provided which includes a predetermined list and associated applications deemed by network **10** as likely to be of interest to the user.

This ability to convert these apparently different search strategies in a single procedure for accessing pre-created library tables is accomplished by translating the procedural elements of the different search techniques into a single set of procedures that will produce a mnemonic; i.e., code word, which can first be searched at the sequence set, described above to identify the object-id for the appropriate library table and, thereafter, enable access of the appropriate table to permit selection of the desired keyword and associated PTO and object-ids. That is to say, the reception system native code simply relates the user-entered character string, alphabetical range, category, or list item of respectively, "Jump", "Index", "Directory", or "Path" to the table codes through the sequence set, so that the appropriate table can be provided to the reception system and application keyword selected. Thus, while the search techniques may appear different to the user, and in fact accommodate the user's preferences and sophistication level, they nonetheless invoke the same efficient procedure of relying upon pre-created searches which identify related application PTOs and object-ids so that the table and objects may be collected and presented at the user's RS **400** where they can be processed, thereby relieving server **205**.

In preferred form, however, in order to enhance presentation speed the Guide operation is specially configured. Rather than relating the keyword mnemonic to a sequence set to identify the table object-id and range of keywords corresponding to the entry PTO and associated object-ids, the Guide operation presents a series of overlapping windows that physically describe the "store" in which shopping is being conducted or the "building" from which information is being provided. The successive windows increase in degree of detail, with the final window presenting a listing of relevant keywords. Further, the PTO and object-ids for the application entry screen are directly related to the graphic presentation of the keywords. This eliminates the need to provide variable fields in the windows for each of the keywords and enables the entry screen to be correlated directly with the window graphic. As will be appreciated, this reduces the number of objects that would otherwise be required to be staged at RS **400** to support pretention of the keyword listing at monitor screen **414**, and thus speeds network response.

A more detailed understanding of the procedure may be had upon a reading of the following description and review of accompanying FIGS. **2**, **3***a* and particularly FIG. **11** which presents a flow diagram for the Jump sequence of the search procedure.

To select a particular partitioned application from among thousands of such applications residing either at the RS **400** or within delivery system **20**, network **10** avoids the need for a user to know or understand, prior to a search, the organization of such partitioned applications and the query techniques necessary to access them. This is accomplished using a collection of related commands, as described below.

In accordance with the invention, the Jump command **296** as seen in FIG. **3***a*, can be selected, by the user from command bar **290**. When Jump command **296** is selected, a window partition **275** is opened. In window **275**, the user is presented and may select from a variety of displayed options that include among others, the Directory command, the Index command, and the Guide command, which when selected, have the effect noted above. Additionally, the user can select a command termed Viewpath which will presents the keywords that currently make up the list of keywords associated with the user's Path command, and from which list the user can select a desired keyword. Still further, and with reference FIG. **11**, which shows the sequence where a user offers a term to identify a subject of interest, the user may enter a keyword at display field **270** within window partition **275** as a "best guess" of the mnemonic character string that is assigned to a partitioned application the user desires (e.g., the user may input such english words as "news," "pet food," "games," etcetera). Where the user enters a character string it is displayed in field **270**, and then searched by RS **400** native code (discussed below) against the sequence sets above noted to identify the object-id for the appropriate table of keywords (not shown) that RS **400**

may request from host **205**. While as noted above, a table may include 10 to 20 keywords, in the preferred embodiment, for the sake of speed and convenience, a typical keyword table includes approximately 12 keywords.

If the string entered by the user matches a keyword existing on one of the keyword tables, and is thus associated with a specific PTO. RS **400** fetches and displays associated objects of the partitioned applications and builds the entry page in accordance with the page composition dictated by the target PTO.

If the string entered by the user does not match a specific keyword. RS **400** presents the user with the option of displaying the table of keywords approximating the specific keyword. The approximate keywords are presented as initialized, cursorable selector fields of the type provided in connection with a Index command. The user may then move the cursor to the nearest approximation of the mnemonic he originally selected, and trigger navigation to the PTO associated with that keyword, navigation being as described hereafter in connection with the RS **400** native code.

If, after selecting the Jump command, the user selects the Index command, RS **400** will retrieve the keyword table residing at RS **400**, and will again build a page with initialized, cursorable fields of keywords. The table fetched upon invoking the Index command will be comprised of alphabetic keywords that occur within the range of the keywords associated with the page template object (PTO) from which the user invoked the Index command. As discussed above, the user may select to navigate to any of this range of PTOs by selecting the relevant keyword from the display. Alternatively, the user can, thereafter, select another range of alphabetical keywords by entering an appropriate character string in a screen field provided or move forward or backward in the collection by selecting the corresponding option.

By selecting the Directory command, RS **400** can be caused to fetch a table of keywords, grouped by categories, to which the PTO of the current partitioned application (as specified by the object set field **630** of the current PEO) belongs. Particularly, by selecting the Directory command, RS **400**, is causes to displays a series of screens each of which contains alphabetically arranged general subject categories from which the user may select. Following selection of a category, a series of keywords associated with the specified category are displayed in further screens together with descriptive statements about the application associated with the keywords. Thereafter, the user can, in the manner previously discussed with regard to the Index command, select from and navigate to the PTOs of keywords which are related to the present page set by subject.

The Guide command provides a navigation method related to a hierarchical organization of applications provided on network **10**, and are described by a series of sequentially presented overlaying windows of a type known in the art, each of which presents an increasing degree of detail for a particular subject area, terminating in a final window that gives keywords associated with the relevant applications. The Guide command makes use of the keyword segment which describes the location of the PTO in a hierarchy (referred to, in the preferred embodiment, as the "BFD," or Building-Floor-Department) as well as an associated keyword character string. The BFD describes the set of menus that are to be displayed on the screen as the sequence of pop-up windows. The Guide command may be invoked by requesting it from the Jump window described above, or by selecting the Menu command on Command Bar **290**. As noted above, in the case of the Guide command, the PTO and object-ids for the application entry screen are directly associated with the graphic of the keyword presented in the final pop-up window. This enables direct access of the application entry screen without need to access the sequence set and keyword table, and thus, reduces response time by reducing the number of objects that must be processed at RS **400**.

Activation of the Path command accesses the user's list of pre-selected keywords without their display, and permits the user to step through the list viewing the respective applications by repeatedly invoking the Path command. As will be appreciated, the user can set a priority for selecting keywords and viewing their associated applications by virtue of where on the list the user places the keywords. More specifically, if the user has several application of particular interest; e.g., news, weather, etc., the user can place them at the top of the list, and quickly step through them with the Path command. Further, the user can view and randomly access the keywords of his list with the Viewpath operation noted above. On activation of Viewpath, the user's Path keywords are displayed and the user can cursor through them in a conventional manner to select a desired one. Further, the user can amend the list as desired by changing the keywords on the list and/or adjusting their relative position. This is readily accomplished by entering the amendments to the list presented at the screen **414** with a series of amendment options presented in a conventional fashion with the list. As noted, the list may be personally selected by the user in the manner described, or created as a default by network **10**.

Collectively, the Jump command, Index command, Directory command, Guide command, and Path command as described enable the user to quickly and easily ascertain the "location" of either the partitioned application presently displayed or the "location" of a desired partitioned application. "Location," as used in reference to the preferred embodiment means the specific relationships that a particular partitioned application bears to other such applications, and the method for selecting particular partitioned applications from such relationships. The techniques for querying a database of objects, embodied in network **10** is an advance over the prior art, insofar as no foreknowledge of either database structure or query technique or syntax is necessary, the structure and search techniques being made manifest to the user in the course of use of the commands.

RS APPLICATION PROTOCOL

RS protocol defines the way the RS supports user application conversation (input and output) and the way RS **400** processes a partitioned application. Partitioned applications are constructed knowing that this protocol will be supported unless modified by the application. The protocol is illustrated FIG. **6**. The boxes in FIG. **6** identify processing states that the RS **400** passes through and the arrows indicate the transitions permitted between the various states and are annotated with the reason for the transition.

The various states are: (A) Initialize RS, (B) Process Objects, (C) Interpretively Execute Pre-processors, (D) Wait for Event, (E) Process Event, and (F) Interpretively Execute Function Extension and/or Post-processors.

The transitions between states are: (1a) Logon Page Template Object Identification (PTO-id), (1b) Object Identification. (2) Trigger Program Object identification (PO-id) & return, (3) Page Partition Template (PPT) or Window Stack Processing complete, (4) Event Occurrence, and (5) Trigger PO-id and Return.

Transition (1a) from Initialize RS (A) to Process Objects (B) occurs when an initialization routine passes the object-id of the logon PTO to object interpreter **435**, when the service is first invoked. Transition (1b) from Process Event (E) to Process Objects (B) occurs whenever a navigation event causes a new page template object identification (PTO-id) to be passed to object interpreter **435**; or when a open window event (verb or function key) occurs passing a window object-id to the object interpreter **435**; or a close window event (verb or function key) occurs causing the current top-most window to be closed.

While in the process object state, object interpreter **435** will request any objects that are identified by external references in call segments. Objects are processed by parsing and interpreting the object and its segments according to the specific object architecture. As object interpreter **435** processes objects, it builds a linked list structure called a page processing table (PPT), shown in FIG. **10**, to reflect the structure of the page, each page partition, Page Element Objects (PEOs) required, program objects (POs) required and each window object (WO) that could be called. Object interpreter **435** requests all objects required to build a page except objects that could be called as the result of some event, such as a HELP window object.

Transition (2) from Process Objects (B) to Interpretively Execute Pre-processors (C) occurs when the object interpreter **435** determines that a pre-processor is to be triggered. Object processor **436** then passes the object-id of the program object to the TBOL interpreter **438**. TBOL interpreter **438** uses the RS virtual machine to interpretively execute the program object. The PO can represent either a selector or an initializer. When execution is complete, a transition automatically occurs back to Process Objects (B).

Selectors are used to dynamically link and load other objects such as PEOs or other PDOs based upon parameters that they are passed when they are called. Such parameters are specified in call segments or selector segments. This feature enables RS **400** to conditionally deliver information to the user base upon predetermined parameters, such as his personal demographics or locale. For example, the parameters specified may be the transaction codes required to retrieve the user's age, sex, and personal interest codes from records contained in user profiles stored at the switch/file server layer **200**.

Initializers are used to set up the application processing environment for a partitioned application and determine what events RS **400** may respond to and what the action will be.

Transition (3) from Process Objects (B) to Wait for Event (D) occurs when object interpreter **435** is finished processing objects associated with the page currently being built or opening or closing a window on a page. In the Wait for Event state (D), an input manager, which in the preferred form shown includes keyboard manager **434** seen in FIG. **8**, accepts user inputs. All keystrokes are mapped from their physical codes to logical keystrokes by the Keyboard Manager **434**, representing keystrokes recognized by the RS virtual machine.

When the cursor is located in a field of a page element, keystrokes are mapped to the field and the partitioned external variable (PEV) specified in the page element object (PEO) field definition segment by the cooperative action of keyboard manager, **434** and display manager **461**. Certain inputs, such as RETURN or mouse clicks in particular fields, are mapped to logical events by keyboard manager **434**, which are called completion (or commit) events. Completion events signify the completion of some selection or specification process associated with the partitioned application and trigger a partition level and/or page level post-processor to process the "action" parameters associated with the user's selection and commit event.

Such parameters are associated with each possible choice or input, and are set up by the earlier interpretive execution of an initializer pre-processor in state (C). Parameters usually specify actions to perform a calculation such as the balance due on an order of several items with various prices using sales tax for the user's location, navigate to PTO-id, open window WO-id or close window. Actions parameters that involve the specification of a page or window object will result in transition (1b) to the Process Objects (B) state after the post-processor is invoked as explained below.

Function keys are used to specify one or more functions which are called when the user strikes these keys. Function keys can include the occurrence of logical events, as explained above. Additionally, certain functions may be "filtered", that is, extended or altered by SET_FUNCTION or TRIGGER_FUNCTION verbs recognized by the RS virtual machine. Function keys cause the PO specified as a parameter of the verb to be interpretively executed whenever that function is called. Applications use this technique to modify or extend the functions provided by the RS.

Transition (5) from Process Event (E) to Interpretively Execute Pre-processors (F) occurs when Process Event State determines that a post-processor or function extension PDO is to be triggered. The id of the program object is then passed to the TBOL interpreter **438**. The TBOL interpreter **438** uses the RS virtual machine to interpretively execute the PO. When execution is complete a transition automatically occurs back to Process Event (E).

RECEPTION SYSTEM SOFTWARE

The reception system **400** software is the interface between the user of personal computer **405** and interactive network **10**. The object of reception system software is to minimize mainframe processing, minimize transmission across the network, and support application extendibility and portability.

RS **400** software is composed of several layers, as shown in FIG. **7**. It includes external software **451**, which is composed of elements well known to the art such as device drivers, the native operating systems; e.g., MS-DOS, machine-specific assembler functions (in the preferred embodiment; e.g., CRC error checking), and "C" runtime library functions; native software **420**; and partitioned applications **410**.

Again with reference to FIG. **7**, native software **420** is compiled from the "C" language into a target machine-specific executable, and is composed of two components: the service software **430** and the operating environment **450**. Operating environment **450** is comprised of the Logical Operating System **432**, or LOS; and a multitasker **433**. Service software **430** provides functions specific to providing interaction between the user and interactive network **10**, while the operating environment **450** provides pseudo multitasking and access to local physical resources in support of service software **430**. Both layers of native software **420** contain kernel, or device independent functions **430** and **432**, and machine-specific or device dependent functions **433**. All device dependencies are in code resident at RS **400**, and are limited to implementing only those functions that are not common across machine types, to enable interactive network **10** to provide a single data stream to all makes of

personal computer which are of the IBM or IBM compatible type. Source code for the native software **420** is included in parent application Ser. No. 388,156 now issued as U.S. Pat. No. 5,347,632, the contents of which patent are incorporated herein by reference. Those interested in a more detailed description of the reception system software may refer to the source code provided in the referenced patent.

Service software **430** is comprised of modules, which are device-independent software components that together obtain, interpret and store partitioned applications existing as a collection of objects. The functions performed by, and the relationship between, the service software **430** module is shown in FIG. **8** and discussed further below.

Through facilities provided by LOS **432** and multitasker **433**, here called collectively operating environment **450**, device-independent multitasking and access to local machine resources, such as multitasking, timers, buffer management, dynamic memory management, file storage and access, keyboard and mouse input, and printer output are provided. The operating environment **450** manages communication and synchronization of service software **430**, by supporting a request/response protocol and managing the interface between the native software **420** and external software **437**.

Applications software layer **410** consists of programs and data written in an interpretive language, "TRINTEX Basic Object Language" or "TBOL," described above. TBOL was written specifically for use in RS **400** and interactive network **10** to facilitate videotext-specific commands and achieve machine-independent compiling. TBOL is constructed as objects, which in interaction with one another comprise partitioned applications.

RS native software **420** provides a virtual machine interface for partitioned applications, such that all objects comprising partitioned applications "see" the same machine. RS native software provides support for the following functions: (1) keyboard and mouse input; (2) text and graphics display; (3) application interpretation; (4) application database management; (5) local application storage; (6) network and link level communications; (7) user activity data collection; and (8) advertisement management.

With reference to FIG. **8**, service software **430** is comprised of the following modules: start-up (not shown); keyboard manger **434**; object interpreter **435**; TBOL interpreter **438**; object storage facility **439**; display manager **461**; data collection manager **441**; ad manager **442**; object/communications manager interface **443**; link communications manager **444**; and fatal error manager **469**. Each of these modules has responsibility for managing a different aspect of RS **400**.

Startup reads RS **400** customization options into RAM, including modem, device driver and telephone number options, from the file CONFIG.SM. Startup invokes all RS **400** component startup functions, including navigation to the first page, a logon screen display containing fields initialized to accept the user's id and password. Since Startup is invoked only at initialization, for simplicity, it has not been shown in FIG. **8**.

The principal function of keyboard manger **434** is to translate personal computer dependent physical input into a consistent set of logical keys and to invoke processors associated with these keys. Depending on the LOS key, and the associated function attached to it, navigation, opening of windows, and initiation of filter or post-processor TBOL programs may occur as the result input events handled by the keyboard manger **434**. In addition, keyboard manger **434** determines inter and intra field cursor movement, and coordinates the display of field text and cursor entered by the user with display manager **461**, and sends information regarding such inputs to data collection manager **441**.

Object interpreter **435** is responsible for building and recursively processing a table called the "Page Processing Table," or PPT. Object interpreter **435** also manages the opening and closing of windows at the current page. Object interpreter **435** is implemented as two sub-components: the object processor **436** and object scanner **437**.

Object processor **436** provides an interface to keyboard manger **434** for navigation to new pages, and for opening and closing windows in the current page. Object processor **436** makes a request to object storage facility **439** for a page template object (PTO) or window object (WO), as requested by keyboard manger **434**, and for objects and their segments which comprise the PTO or WO returned by object storage facility **439** to object processor **436**. Based on the particular segments comprising the object(s) making up the new PTO or WO, object processor **436** builds or adds to the page processing table (PPT), which is an internal, linked-list, global data structure reflecting the structure of the page or page format object (PFO), each page partition or page element object (PEO), and program objects (POs) required and each window object (WO) that could be called. Objects are processed by parsing and interpreting each object and its segment(s) according to their particular structure as formalized in the data object architecture (DOA). While in the process object state, (state "B" of FIG. **6**), object processor **436** will request any objects specified by the PTO that are identified by external references in call segments (e.g. field level program call **518**, page element selector call **524**, page format call **526** program call **532**, page element call **522** segments) of such objects, and will, through a request to TBOL interpreter **438**, fire initializers and selectors contained in program data segments of all PTO constituent program objects, at the page, element, and field levels. Object processor **436** requests all objects required to build a page, except objects that could only be called as the result of some event external to the current partitioned application, such as a HELP window object. When in the course of building or adding to the PPT and opening/closing WOs, object processor encounters a call to an "ADSLOT" object id, the next advertisement object id at ad manager **442** is fetched, and the identified advertisement object is retrieved either locally, if available, or otherwise from the network, so that the presentation data for the advertisement can be sent to display manager **461** along with the rest of the presentation data for the other objects to enable display to the user. Object processor **436** also passes to data collection manager **441** all object ids that were requested and object ids that were viewed. Upon completion of page or window processing, object processor **436** enters the wait for event state, and control is returned to keyboard manger **434**.

The second component of object interpreter **435**, object scanner **437**, provides a file-like interface, shared with object storage facility **439**, to objects currently in use at RS **400**, to enable object processor **436** to maintain and update the PPT. Through facilities provided by object scanner **437**, object processor recursively constructs a page or window in the requested or current partitioned application, respectively.

Object storage facility **439** provides an interface through which object interpreter **435** and TBOL interpreter **438** either synchronously request (using the TBOL verb operator "GET") objects without which processing in either module cannot continue, or asynchronously request (using the TBOL verb operator "FETCH") objects in anticipation of

later use. Object storage facility **439** returns the requested objects to the requesting module once retrieved from either local store **440** or interactive network **10**. Through control structures shared with the object scanner **437**, object storage facility determines whether the requested object resides locally, and if not, makes an attempt to obtain it from interactive network **10** through interaction with link communications manager **444** via object/communications manager interface **443**.

When objects are requested from object storage facility **439**, only the latest version of the object will be provided to guarantee currency of information to the user. Object storage facility **439** assures currency by requesting version verification from network **10** for those objects which are available locally and by requesting objects which are not locally available from delivery system **20** where currency is maintained.

Version verification increases response time. Therefore, not all objects locally available are version checked each time they are requested. Typically, objects are checked only the first time they are requested during a user session. However, there are occasions, as for example in the case of objects relating to news applications, where currency is always checked to assure integrity of the information.

The frequency with which the currency of objects is checked depends on factors such as the frequency of updating of the objects. For example, objects that are designated as ultrastable in a storage control parameter in the header of the object are never version checked unless a special version control object sent to the RS as part of logon indicates that all such objects must be version checked. Object storage facility **439** marks all object entries with such a stability category in all directories indicating that they must be version checked the next time they are requested.

Object storage facility **439** manages objects locally in local store **440**, comprised of a cache (segmented between available RAM and a fixed size disk file), and stage (fixed size disk file). Ram and disk cached objects are retained only during user sessions, while objects stored in the stage file are retained between sessions. The storage control field, located in the header portion of an object, described more fully hereafter as the object "storage candidacy", indicates whether the object is stageable, cacheable or trashable.

Stageable objects must not be subject to frequent change or update. They are retained between user sessions on the system, provided storage space is available and the object has not discarded by a least-recently-used (LRU) algorithm of a conventional type; e.g., see *Operating System Theory*, by Coffman, Jr. and Denning, Prentice Hall Publishers, New York, 1973, which operates in combination with the storage candidacy value to determine the object storage priority, thus rendering the stage self-configuring as described more fully hereafter. Over time, the self-configuring stage will have the effect of retaining within local disk storage those objects which the user has accessed most often. The objects retained locally are thus optimized to each individual user's usage of the applications in the system. Response time to such objects is optimized since they need not be retrieved from the interactive computer system.

Cacheable objects can be retained during the current user session, but cannot be retained between sessions. These objects usually have a moderate update frequency. Object storage facility **439** retains objects in the cache according to the LRU storage retention algorithm. Object storage facility **439** uses the LRU algorithm to ensure that objects that are least frequently used forfeit their storage to objects that are more frequently used.

Trashable objects can be retained only while the user is in the context of the partitioned application in which the object was requested. Trashable objects usually have a very high update frequency and must not be retained to ensure that the user has access to the most current data.

More particularly and, as noted above, in order to render a public informational and transactional network of the type considered here attractive, the network must be both economical to use and fast. That is to say, the network must supply information and transactional support to the user at minimal costs and with a minimal response time. These objectives are sought to be achieved by locating as many information and transactional support objects which the user is likely to request, as close to the user as possible; i.e., primarily at the user's RS **400** and secondarily at delivery system **20**. In this way, the user will be able to access objects required to support a desired application with minimal intervention of delivery system **20**, thus reducing the cost of the session and speeding the response time.

However, the number of objects that can be maintained at RS **400** is restricted by at least two factors: the RS **400** storage capacity; i.e., RAM and disk sizes, and the need to maintain the stored objects current.

In order to optimize the effectiveness of the limited storage space at RS **400**, the collection of objects is restricted to those likely to be requested by the user; i.e., tailored to the user's tastes—and to those least likely to be time sensitive; i.e., objects which are stable. To accomplish this, objects are coded for storage candidacy to identify when they will be permitted at RS **400**, and subject to the LRU algorithm to maintain presence at RS **400**. Additionally, to assure currency of the information and transaction support provided at RS **400**, objects are further coded for version identification and checking in accordance with a system of priorities that are reflected in the storage candidacy coding.

Specifically, to effect object storage management, objects are provided with a coded version id made up of the storage control byte and version control bytes identified above as elements of the object header, specifically, bytes **16** and **18** shown in FIG. **4***b*. In preferred form, the version id is comprised of bytes **16** and **18** to define two fields, a first 13 bit field to identify the object version and a second three bite field to identify the object storage candidacy.

In this arrangement, the storage candidacy value of the object is addressed to not only the question of storage preference but also object currency. Specifically, the storage candidacy value establishes the basis upon which the object will be maintained at RS **400** and also identifies the susceptibility of the object to becoming stale by dictating when the object will be version checked to determine currency.

The version value of the object on the other hand, provides a parameter that can be checked against predetermined values available from delivery system **20** to determine whether an object stored at RS **400** is sufficiently current to permit its continued use, or whether the object has become stale and needs to be replaced with a current object from delivery system **20**.

Still further, object storage management procedure further includes use of the LRU algorithm, for combination with the storage and version coding to enable discarding of objects which are not sufficiently used to warrant retention, thus personalizing the store of objects at RS **400** to the user's tastes. Particularly, object storage facility **439**, in accordance with the LRU algorithm maintains a usage list for objects. As objects are called to support the user's applications

requests, the objects are moved to the top of a usage list. As other objects are called, they push previously called objects down in the list. If an object is pushed to the bottom of the list before being recalled, it will be forfeited from the list if necessary to make room for the next called object. As will be appreciated, should a previously called object be again called before it is displaced from the list, it will be promoted to the top of the list, and once more be subject to depression in the list and possible forfeiture as other objects are called.

As pointed out above, in the course of building the screens presented to the user, objects will reside at various locations in RS **400**. For example, objects may reside in the RS **400** RAM where the object is supporting a particular application screen then running or in a cache maintained at either RAM or disk **424** where the object is being held for an executing application or staged on the fixed size file on disk **424** noted above where the object is being held for use in application likely to be called by the user in the future.

In operation, the LRU algorithm is applied to all these regions and serves to move an object from RAM cache to disk cache to disk file, and potentially off RS **400** depending on object usage.

With regard to the storage candidacy value, in this arrangement, the objects stored at RS **400** include a limited set of permanent objects; e.g., those supporting logon and logoff, and other non-permanent objects which are subject to the LRU algorithm to determine whether the objects should be forfeited from RS **400** as other objects are added. Thus, in time, and based on the operation of the LRU algorithm and the storage candidacy value, the collection of objects at RS **400** will be tailored to the usage characteristics of the subscriber; i.e., self-configuring.

More particularly, the 3-bit field of the version id that contains the storage candidacy parameter can have 8 different values. A first candidacy value is applied where the object is very sensitive to time; e.g., news items, volatile pricing information such as might apply to stock quotes, etc. In accordance with this first value, the object will not be permitted to be stored on RS **400**, and RS **400** will have to request such objects from delivery system **20** each time it is accessed, thus, assuring currency. A second value is applied where the object is sensitive to time but less so than the first case; e.g., the price of apples in a grocery shopping application. Here, while the price might change from day to day, it is unlikely to change during a session. Accordingly the object will be permitted to persist in RAM or at the disk cache during a session, but will not be permitted to be maintained at RS **400** between sessions.

Continuing down the hierarchy of time sensitivity, where the object concerns information sufficiently stable to be maintained between sessions, a third storage candidacy value is set to permit the object to be stored at RS **400** between sessions, on condition that the object will be version check the first time it is accessed in a subsequent session. As will be appreciated, during a session, and under the effect of the LRU algorithm, lack of use at RS **400** of the object may result in it being forfeited entirely to accommodate new objects called for execution at RS **400**.

Still further, a fourth value of storage candidacy is applied where the object is considered sufficiently stable as not to require version checking between sessions; e.g., objects concerning page layouts not anticipated to change. In this case, the storage candidacy value may be encoded to permit the object to be retained from session to session without version checking. Here again, however, the LRU algorithm may cause the object to forfeit its storage for lack of use.

Where the object is of a type required to be stored at RS **400**, as for example, objects needed to support standard screens, it is coded for storage between sessions and not subject to the LRU algorithm forfeiture. However, where such objects are likely to change in the future they may be required to be version checked the first time they are accessed in a session and thus be given a fifth storage candidacy value. If, on the other hand, the required stored object is considered likely to be stable and not require even version checking; e.g., logon screens, it will be coded with a sixth storage candidacy value for storage without version checking so as to create a substantially permanent object.

Continuing, where a RS **400** includes a large amount of combined RAM and disk capacity, it would permit more objects to be stored. However, if objects were simply coded in anticipation of the larger capacity, the objects would potentially experience difficulty, as for example, undesired forfeiture due to capacity limitations if such objects were supplied to RS **400** units having smaller RAM and disk sizes. Accordingly, to take advantage of the increased capacity of certain RS **400** units without creating difficulty in lower capacity units, objects suitable for storage in large capacity units can be so coded for retention between sessions with a seventh and eighth storage candidacy value depending upon whether the stored large capacity object requires version checking or not. Here, however, the coding will be interpreted by smaller capacity units to permit only cacheable storage to avoid undesirable forfeiture that might result from over filling the smaller capacity units.

Where an object is coded for no version checking need may nonetheless arise for a version check at some point. To permit version checking of such objects, a control object is provided at RS **400** that may be version checked on receipt of a special communication from delivery system **20**. If the control object fails version check, then a one shot version checking attribute is associated with all existing objects in RS **400** that have no version checking attributes. Thereafter, the respective objects are version checked, the one shot check attribute is removed and the object is caused to either revert to its previous state if considered current or be replaced if stale.

Still further, objects required to be stored at RS **400** which are not version checked either because of lack of requirement or because of no version check without a control object, as described above, can accumulate in RS **400** as dead objects. To eliminate such accumulation, all object having required storage are version checked over time. Particularly, the least recently used required object is version checked during a session thus promoting the object to the top of the usage list if it is still to be retained at RS **400**. Accordingly, one such object will be checked per session and over time, all required objects will be version checked thereby eliminating the accumulation of dead objects.

However, in order to work efficiently, the version check attribute of the object should be ignored, so that even required object can be version checked. Yet, in certain circumstances, e.g., during deployment of new versions of the reception system software containing new objects not yet supported on delivery system **20** which may be transferred to the fixed storage file of RS **400** when the new version is loaded, unconditional version checking may prematurely deletes the object from the RS **400** as not found on delivery system **20**. To avoid this problem, a sweeper control segment in the control object noted above can be used to act as a switch to turn the sweep of dead objects on and off.

With respect to version checking for currency, where an object stored at RS **400** is initially fetched or accessed during

a session, a request to delivery system **20** is made for the object by specifying the version id of the object stored at RS **400**.

In response, delivery system **20** will advise the reception system **400** either that the version id of the stored object matches the currency value; i.e., the stored object is acceptable, or deliver a current object that will replace the stored object shown to be stale. Alternatively, the response may be that the object was not found. If the version of the stored object is current, the stored object will be used until verified again in accordance with its storage candidacy. If the stored object is stale, the new object delivered will replace the old one and support the desired screen. If the response is object not found, the stored object will be deleted.

Therefore, based on the above description, network **10** is seen to include steps for execution at storage facility **439** which enables object reception, update and deletion by means of a combination of operation of the LRU algorithm and interpretation of the storage candidacy and version control values. In turn, these procedures cooperate to assure a competent supply of objects at RS **400** so as to reduce the need for intervention of delivery system **20**, thus reducing cost of information supply and transactional support so as to speed the response to user requests.

TBOL interpreter **438** shown in FIG. **8** provides the means for executing program objects, which have been written using an interpretive language, TBOL described above. TBOL interpreter **438** interprets operators and operand contained in program object **508**, manages TBOL variables and data, maintains buffer and stack facilities, and provides a runtime library of TBOL verbs.

TBOL verbs provide support for data processing, program flow control, file management, object management, communications, text display, command bar control, open/close window, page navigation and sound. TBOL interpreter also interacts with other native modules through commands contained in TBOL verbs. For example: the verb "navigate" will cause TBOL interpreter **438** to request object interpreter **435** to build a PPT based on the PTO id contained in the operand of the NAVIGATE verb; "fetch" or "GET" will cause TBOL interpreter **438** to request an object from object storage facility **439**; "SET_FUNCTION" will assign a filter to events occurring at the keyboard manger **434**; and "FORMAT," "SEND," and "RECEIVE" will cause TBOL interpreter **438** to send application level requests to object/communications manager interface **433**.

Data areas managed by TBOL interpreter **438** and available to TBOL programs are Global External Variables (GEVs), Partition External Variables (PEVs), and Runtime Data Arrays (RDAs).

GEVs contain global and system data, and are accessible to all program objects as they are executed. GEVs provide a means by which program objects may communicate with other program objects or with the RS native code, if declared in the program object. GEVs are character string variables that take the size of the variables they contain. GEVs may preferably contain a maximum of 32,000 variables and are typically used to store such information as program return code, system date and time, or user sex or age. TBOL interpreter **438** stores such information in GEVs when requested by the program which initiated a transaction to obtain these records from the RS or user's profile stored in the interactive system.

Partition external variables (PEVs) have a scope restricted to the page partition on which they are defined. PEVs are used to hold screen field data such that when PEOs and window objects are defined, the fields in the page partitions with which these objects are to be associated are each assigned to a PEV. When applications are executed, TBOL interpreter **438** transfers data between screen fields and their associated PEV. When the contents of a PEV are modified by user action or by program direction, TBOL interpreter **428** makes a request to display manager **461** to update the screen field to reflect the change. PEVs are also used to hold partition specific application data, such as tables of information needed by a program to process an expected screen input.

Because the scope of PEVs is restricted to program objects associated with the page partition in which they are defined, data that is to be shared between page partitions or is to be available to a page-level processor must be placed in GEVs or RDAs.

RDAs are internal stack and save buffers used as general program work areas. RDAs are dynamically defined at program object "runtime" and are used for communication and transfer of data between programs when the data to be passed is not amenable to the other techniques available. Both GEVs and RDAs include, in the preferred embodiment, 8 integer registers and 8 decimal registers. Preferably, there are also 9 parameter registers limited in scope to the current procedure of a program object.

All variables may be specified as operand of verbs used by the virtual machine. The integer and decimal registers may be specified as operand for traditional data processing. The parameter registers are used for passing parameters to "called" procedures. The contents of these registers are saved on an internal program stack when a procedure is called, and are restored when control returns to the "calling" procedure from the "called" procedure.

TBOL interpreter **438**, keyboard manger **434**, object interpreter **435**, and object storage facility **439**, together with device control provided by operating environment **450**, have principal responsibility for the management and execution of partitioned applications at the RS **400**. The remaining native code modules function in support and ancillary roles to provide RS **400** with the ability display partitioned applications to the user (display manager **461**), display advertisements (ad manager **442**), to collect usage data for distribution to interactive network **10** for purposes of targeting such advertisements (data collection manager **441**), and prepare for sending, and send, objects and messages to interactive network **10** (object/communications manager interface **443** and link communications manager **444**) Finally, the fatal error manager exists for one purpose: to inform the user of RS **400** and transmit to interactive network **10** the inability of RS **400** to recover from a system error.

Display manager **461** interfaces with a decoder using the North American Presentation Level Protocol Syntax (NAPLPS), a standard for encoding graphics data, or text code, such as ASCII, which are displayed on monitor **412** of the user's personal computer **405** as pictorial codes. Codes for other presentation media, such as audio, can be specified by using the appropriate type code in the presentation data segments. Display manager **461** supports the following functions: send NAPLPS strings to the decoder; echo text from a PEV; move the cursor within and between fields; destructive or non-destructive input field character deletion; "ghost" and "unghost" fields (a ghosted field is considered unavailable, unghosted available); turn off or on the current field cursor; open, close, save and restore bit maps for a

graphics window; update all current screen fields by displaying the contents of their PEVs, reset the NAPLPS decoder to a known state; and erase an area of the screen by generating and sending NAPLPS to draw a rectangle over that area. Display manager **461** also provides a function to generate a beep through an interface with a machine-dependent sound driver.

Ad manager **442** is invoked by object interpreter **435** to return the object id of the next available advertisement to be displayed. Ad manager **442** maintains a queue of advertising object id's targeted to the specific user currently accessing interactive network **10**. Advertising objects are pre-fetched from interactive system **10** from a personalized queue of advertising ids that is constructed using data previously collected from user generated events and/or reports of objects used in the building of pages or windows, compiled by data collection manager **466** and transmitted to interactive system **10**.

Advertising objects **510** are PEOs that, through user invocation of a "LOOK" command, cause navigation to partitioned applications that may themselves support, for example, ordering and purchasing of merchandise.

An advertising object id list, or "ad queue," is requested in a transaction message to delivery system **20** by ad manager **442** immediately after the initial logon response. The logon application at RS **400** places the advertising list in a specific RS global storage area called a SYS_GEV (system global external variable), which is accessible to all applications as well as to the native RS code). The Logon application also obtains the first two ad object id's from the queue and provides them to object storage facility **439** so the advertising objects can be requested. However, at logon, since no advertising objects are available at RS local storage facilities **440**, ad objects, in accordance with the described storage candidacy, not being retained at the reception system between sessions, they must be requested from interactive network **10**.

In accordance with the preferred form of network **10**, the following parametric values are established for ad manager **442**: advertising object is queue capacity, replenishment threshold for advertising object id's and replenishment threshold for number of outstanding pre-fetched advertising objects. These parameters are set up in GEVs of the RS virtual machine by the logon application program object from the logon response from high function system **110**. The parameters are then also accessible to the ad manager **442**. Preferred values are an advertising queue capacity of 15, replenishment value of 10 empty queue positions and a pre-fetched advertising object threshold of 3.

Ad manager **442** pre-fetches advertising objects by passing advertising object id's from the advertising queue to object storage facility **439** which then retrieves the object from the interactive system if the object is not available locally. Advertising objects are pre-fetched, so they are available in RS local store **440** when requested by object interpreter **435** as it builds a page. The ad manager **442** pre-fetches additional advertising objects whenever the number of pre-fetched advertising objects not called by object interpreter **435**; i.e. the number of remaining advertising objects, falls below the pre-fetch advertising threshold.

Whenever the advertising object id queue has more empty positions than replenishment threshold value, a call is made to the advertising object id queue application in high function system **110** shown in FIG. **2**, via object/communications manager interface **443** for a number of advertising object id's equal to the threshold value. The response message from system **110** includes a list of advertising object id's, which ad manager **442** enqueues.

Object interpreter **435** requests the object id of the next advertising object from ad manager **442** when object interpreter **435** is building a page and encounters an object call for a partition and the specified object-id equals the code word, "ADSLOT." If this is the first request for an advertising object id that ad manager **442** has received during this user's session, ad manager **442** moves the advertising object id list from the GEV into its own storage area, which it uses as an advertising queue and sets up its queue management pointers, knowing that the first two advertising objects have been pre-fetched.

Ad manager **442** then queries object storage facility **439**, irrespective of whether it was the first request of the session. The query asks if the specified advertising object id pre-fetch has been completed, i.e., is the object available locally at the RS. If the object is available locally, the object-id is passed to object interpreter **435**, which requests it from object storage facility **439**. If the advertising object is not available in local store **440**, ad manager **442** attempts to recover by asking about the next ad that was pre-fetched. This is accomplished by swapping the top and second entry in the advertising queue and making a query to object storage facility **439** about the new top advertising object id. If that object is not yet available, the top position is swapped with the third position and a query is made about the new top position.

Besides its ability to provide advertising that have been targeted to each individual user, two very important response time problems have been solved by ad manager **442**. The first is to eliminate from the new page response time the time it takes to retrieve an advertising object from the host system. This is accomplished by using the aforementioned pre-fetching mechanism.

The second problem is caused by pre-fetching, which results in asynchronous concurrent activities involving the retrieval of objects from interactive system **10**. If an advertising object is pre-fetched at the same time as other objects required for a page are requested, the transmission of the advertising object packets could delay the transmission of the other objects required to complete the current page by the amount of time required to transmit the advertising object(s). This problem is solved by the structuring the requests from object interpreter **435** to the ad manager **442** in the following way:

1. Return next object id of pre-fetched advertising object & pre-fetch another;
2. Return next advertising object id only; and
3. Pre-fetch next advertising object only.

By separating the function request (1) into its two components, (2) and (3), object interpreter **435** is now able to determine when to request advertising object id's and from its knowledge of the page build process, is able to best determine when another advertising object can be pre-fetched, thus causing the least impact on the page response time. For example, by examining the PPT, object interpreter **435** may determine whether any object requests are outstanding. If there are outstanding requests, advertising request type 2 would be used. When all requested objects are retrieved, object interpreter **435** then issues an advertising request type 3. Alternatively, if there are no outstanding requests, object interpreter **435** issues an advertising request type 1. This typically corresponds to the user's "think time" while examining the information presented and when RS **400** is in the Wait for Event state (D).

Data collection manager **441** is invoked by object interpreter **435** and keyboard manger **434** to keep records about what objects a user has obtained (and, if a presentation data segment **530** is present, seen) and what actions users have taken (e.g. "NEXT," "BACK," "LOOK," etc.)

The data collection events that are to be reported during the user's session are sensitized during the logon process. The logon response message carries a data collection indicator with bit flags set to "on" for the events to be reported. These bit flags are enabled (on) or disabled (off) for each user based on information contained in the user's profile stored and sent from high function host **110**. A user's data collection indicator is valid for the duration of his session. The type of events to be reported can be changed at will in the host data collection application. However, such changes will affect only users who logon after the change.

Data collection manager **441** gathers information concerning a user's individual system usage characteristics. The types of informational services accessed, transactions processed, time information between various events, and the like are collected by data collection manager **441**, which compiles the information into message packets (not shown). The message packets are sent to network **10** via object/communication manager interface **443** and link communications manager **444**. Message packets are then stored by high function host **110** and sent to an offline processing facility for processing. The characteristics of users are ultimately used as a means to select or target various display objects, such as advertising objects, to be sent to particular users based on consumer marketing strategies, or the like, and for system optimization.

Object/communications manager interface **443** is responsible for sending and receiving DIA (Data Interchange Architecture described above) formatted messages to or from interactive network **10**. Object/communications manager **443** also handles the receipt of objects, builds a DIA header for messages being sent and removes the header from received DIA messages or objects, correlates requests and responses, and guarantees proper block sequencing. Object/communications manager interface **443** interacts with other native code modules as follows: object/communications manager **443** (1) receives all RS **400** object requests from object storage facility **439**, and forwards objects received from network **10** via link communications manager **444** directly to the requesting modules; (2) receives ad list requests from ad manager **442**, which thereafter periodically calls object/communications manager **443** to receive ad list responses; (3) receives data collection messages and send requests from data collection manager **441**; (4) receives application-level requests from TBOL interpreter **438**, which also periodically calls object/communications manager interface **443** to receive responses (if required); and (5) receives and sends DIA formatted objects and messages from and to link communications manager **444**.

Object/communications manager interface **443** sends and receives DIA formatted messages on behalf of TBOL interpreter **438** and sends object requests and receives objects on behalf of object storage facility **439**. Communication packets received containing parts of requested objects are passed to object storage facility **439** which assembles the packets into the object before storing it. If the object was requested by object interpreter **435**, all packets received by object storage facility **439** are also passed to object interpreter **435** avoiding the delay required to receive an entire object before processing the object. Objects which are pre-fetched are stored by object storage facility **439**.

Messages sent to interactive network **10** are directed via DIA to applications in network **10**. Messages may include transaction requests for records or additional processing of records or may include records from a partitioned application program object or data collection manager **441**. Messages to be received from network **10** usually comprise records requested in a previous message sent to network **10**. Requests received from object storage facility **439** include requests for objects from storage in interactive system **10**. Responses to object requests contain either the requested object or an error code indicating an error condition.

Object/communications manager **443** is normally the exclusive native code module to interface with link communications manager **444** (except in the rare instance of a fatal error). Link communications manager **444** controls the connecting and disconnecting of the telephone line, telephone dialing, and communications link data protocol. Link communications manager **444** accesses network **10** by means of a communications medium (not shown) link communications manager **444**, which is responsible for a dial-up link on the public switched telephone network (PSTN). Alternatively, other communications means, such as cable television or broadcast media, may be used. Link communications manager **444** interfaces with TBOL interpreter for connect and disconnect, and with interactive network **10** for send and receive.

Link communications manager **444** is subdivided into modem control and protocol handler units. Modem control (a software function well known to the art) hands the modem specific handshaking that occurs during connect and disconnect. Protocol handler is responsible for transmission and receipt of data packets using the TCS (TRINTEX Communications Subsystem) protocol (which is a variety of OSI link level protocol, also well known to the art).

Fatal error manager **469** is invoked by all reception system components upon the occurrence of any condition which precludes recovery. Fatal error manager **469** displays a screen to the user with a textual message and an error code through display manager **461**. Fatal error manager **469** sends an error report message through the link communications manager **444** to a subsystem of interactive network **10**.

The source code for the reception system software as noted above is described in parent application Ser. No. 388,156 filed Jul. 28, 1989, now issued as U.S. Pat. No. 5,347,632, the contents of which are incorporated herein by reference.

SAMPLE APPLICATION

Page **255** illustrated in FIG. **3***b* corresponds to a partitioned application that permit's a user to purchase apples. It shows how the monitor screen **414** of the reception system **400** might appear to the user. Displayed page **255** includes a number of page partitions and corresponding page elements.

The page template object (PTO) **500** representing page **255** is illustrated in FIG. **9**. PTO **500** defines the composition of the page, including header **250**, body **260**, display fields **270**, **271**, **272**, advertising **280**, and command bar **290**. Page element objects (PEOs) **504** are associated with page partitions numbered; e.g., **250**, **260**, **280**. They respectively, present information in the header **250**, identifying the page topic as ABC APPLES; in the body **260**, identifying the cost of apples; and prompt the user to input into fields within body **260** the desired number of apples to be ordered. In advertising **280**, presentation data and a field representing a post-processor that will cause the user to navigate to a targetable advertising, is presented.

In FIG. **9**, the structure of PTO **500** can be traced. PTO **500** contains a page format call segment **526**, which calls

page format object (PFO) **502**. PFO **502** describes the location and size of partitions on the page and numbers assigned to each partition. The partition number is used in page element call segments **522** so that an association is established between a called page element object (PEO) **504** and the page partition where it is to be displayed. Programs attached to this PEO can be executed only when the cursor is in the page partition designated within the PEO.

PTO **500** contains two page element call segments **522**, which reference the PEOs **504** for partitions **250** and **260**. Each PEO **504** defines the contents of the partition. The header in partition **250** has only a presentation data segment **530** in its PEO **504**. No input, action, or display fields are associated with that partition.

The PEO **504** for partition **260** contains a presentation data segment **530** and field definition segments **516** for the three fields that are defined in that partition. Two of the fields will be used for display only. One field will be used for input of user supplied data.

In the example application, the PEO **504** for body partition **260** specifies that two program objects **508** are part of the body partition. The first program, shown in Display field **270**, **271**, **272**, is called an initializer and is invoked unconditionally by TBOL interpreter **438** concurrently with the display of presentation data for the partition. In this application, the function of the initializer is represented by the following pseudo-code:

1. Move default values to input and display fields;
2. "SEND" a transaction to the apple application that is resident on interactive system **10**;
3. "RECEIVE" the result from interactive system **10**; i.e. the current price of an apple;
4. Move the price of an apple to PEV **271** so that it will be displayed;
5. Position the cursor on the input field; and
6. Terminate execution of this logic.

The second program object **508** is a field post-processor. It will be invoked conditionally, depending upon the user keystroke input. In this example, it will be invoked if the user changes the input field contents by entering a number. The pseudo code for this post-processor is as follows:

1. Use the value in PEV **270** (the value associated with the data entered by the user into the second input data field **270**) to be the number of apples ordered.
2. Multiply the number of apples ordered times the cost per apple previously obtained by the initializer;
3. Construct a string that contains the message "THE COST OF THE APPLES YOU ORDERED IS $45.34;";
4. Move the string into PEV **272** so that the result will be displayed for the user; and
5. Terminate execution of this logic.

The process by which the "APPLES" application is displayed, initialized, and run is as follows.

The "APPLES" application is initiated when the user navigates from the previous partitioned application, with the navigation target being the object id of the "APPLES" PTO **500** (that is, object id ABC1). This event causes keyboard manager **434** to pass the PTO object id, ABC1 (which may, for example, have been called by the keyword navigation segment **520** within a PEO **504** of the previous partitioned application), to object interpreter **435**. With reference to the RS application protocol depicted in FIG. **6**, when the partitioned application is initiated, RS **400** enters the Process Object state (B) using transition (1). Object interpreter **435** then sends a synchronous request for the PTO **500** specified in the navigation event to object storage facility **439**. Object storage facility **439** attempts to acquire the requested object from local store **440** or from delivery system **20** by means of object/communication manager **443**, and returns an error code if the object cannot be acquired.

Once the PTO **500** is acquired by object/communications manager **443**, object interpreter **435** begins to build PPT by parsing PTO **500** into its constituent segment calls to pages and page elements, as shown in FIG. **4***d* and interpreting such segments. PFO and PEO call segments **526** and **522** require the acquisition of the corresponding objects with object id's <ABCF>, <ABCX> and <ABCY>. Parsing and interpretation of object ABCY requires the further acquisition of program objects <ABCI> and <ABCJ>.

During the interpretation of the PEOs **504** for partitions **250** and **260**, other RS **400** events are triggered. This corresponds to transition (2) to interpret pre-processors state (C) in FIG. **6**. Presentation data **530** is sent to display manager **461** for display using a NAPLPS decoder within display manager **461**, and, as the PEO <ABCY> for partition **260** is parsed and interpreted by object interpreter **435**, parameters in program call segment **532** identify the program object <ABCI> as an initializer. Object interpreter **435** obtains the program object from object storage facility **439**, and makes a request to TBOL interpreter **438** to execute the initializer program object **508** <ABCI>. The initializer performs the operations specified above using facilities of the RS virtual machine. TBOL interpreter **438**, using operating environment **450**, executes initializer program object **506** <ABCI>, and may, if a further program object **508** is required in the execution of the initializer, make a synchronous application level object request to object storage facility **439**. When the initializer terminates, control is returned to object interpreter **435**, shown as the return path in transition (2) in FIG. **6**.

Having returned to the process object state (B), object processor **435** continues processing the objects associated with PTO <ABC1>. Object interpreter continues to construct the PPT, providing RS **400** with an environment for subsequent processing of the PTO <ABC1> by pre-processors and post-processors at the page, partition, and field levels. When the PPT has been constructed and the initializer executed, control is returned to keyboard manager **434**, and the RS enters the wait for event (E) State, via transition (4), as shown in FIG. **6**.

In the wait for event state, the partitioned application waits for the user to create an event. In any partitioned application, the user has many options. For example, the user may move the cursor to the "JUMP" field **296** on the command bar **290**, which is outside the current application, and thus cause subsequent navigation to another application. For purposes of this example, it is assumed that the user enters the number of apples he wishes to order by entering a digit in display field **271**.

Keyboard manager **434** translates the input from the user's keyboard to a logical representation independent of any type of personal computer. Keyboard manager **434** saves the data entered by the user in a buffer associated with the current field defined by the location of the cursor. The buffer is indexed by its PEV number, which is the same as the field number assigned to it during the formation of the page element. Keyboard manager **434** determines for each keystroke whether the keystroke corresponds to an input event or to an action or completion event. Input events are logical keystrokes and are sent by keyboard manager to display manager **461**, which displays the data at the input

field location. Display manager **461** also has access to the field buffer as indexed by its PEV number.

The input data are available to TBOL interpreter **438** for subsequent processing. When the cursor is in a partition, only the PEVs for that partition are accessible to the RS virtual machine. After the input from the user is complete (as indicated by a user action such as pressing the RETURN key or entry of data into a field with an action attribute), RS **400** enters the Process Event state (E) via transition (4).

For purposes of this example, let us assume that the user enters the digit "5" in input field **270**. A transition is made to the process event state (E). Keyboard manager **434** and display manager **437** perform a number of actions, such as the display of the keystroke on the screen, the collection of the keystroke for input, and optionally, the validation of the keystroke, i.e. numeric input only in numeric fields. When the keystroke is processed, a return is made to the wait for event state (D). Edit attributes are specified in the field definition segment.

Suppose the user inputs a "6" next. A transition occurs to the PE state and after the "6" is processed, the Wait for Event (D) state is reentered. If the user hits the "completion" key (e.g., ENTER) the Process Event (E) state will be entered. The action attributes associated with field **272** identify this as a system event to trigger post-processor program object <ABCJ>. When the interpretive execution of program object <ABCJ> is complete, the wait for event state (D) will again be entered. The user is then free to enter another value in the input field, or select a command bar function and exit the apples application.

While this invention has been described in its preferred form, it will be appreciated that changes may be made in the form, construction, procedure and arrangement of its various elements and steps without departing from its spirit or scope.

What we claim is:

1. A method for presenting interactive applications on a computer network, the network including a multiplicity of user reception systems at which respective users may request a multiplicity of available applications, the respective reception systems including a monitor at which the applications requested can be presented as one or more screens of display, the method comprising the steps of:

a. generating a screen display at a respective reception system for a requested application, the screen display being generated by the respective reception system from data objects having a prescribed data structure, at least some of which objects may be stored at the respective reception system, the screen display including a plurality of partitions, the partitions being constructed from objects, the objects being retrieved from the objects stored at the respective reception system, or if unavailable from the objects stored at the respective reception system, then from the network, such that at least some of the objects may be used in more than one application;

b. generating at least a first partition for presenting applications; and

c. generating concurrently with the first partition at least a second partition for presenting a plurality of command functions, the command functions including at least a first group which are selectable to permit movement between applications.

2. The method of claim **1** wherein the data structure of the objects includes a header and one or more data segments and wherein generating the second partition includes providing the first group of command functions with a first subgroup of command functions which are selectable to permit random movement between applications.

3. The method of claim **2** wherein the objects are stored at the respective reception systems in accordance with a predetermined plan, and wherein providing the first subgroup of commands includes providing a command for causing the user to be presented with a t least one procedure for navigating to a new application.

4. The method of claim **2** wherein the predetermined plan for storing objects at the respective reception systems includes providing the objects with a storage control parameter in their respective headers, and wherein providing the first subgroup of command functions includes providing at least one command for causing the user to be presented with a plurality of different procedure for navigating to a new application.

5. The method of claim **4** wherein the object storage controle parameter is dependent on the currency of the object data, and wherein providing the navigation procedures includes enabling the user to enter a character string at the reception system to randomly search the available applications for a desired application.

6. The method of claim **4** wherein providing the navigation procedures includes enabling the user to access an index of available applications from which a desired application may be selected.

7. The method of claim **4** wherein providing the navigation procedures includes enabling the user to access a directory of application subject matter from which a desired application may be selected.

8. The method of claim **4** wherein providing the navigation procedures includes enabling the user to access a physical analogy of the available applications from which a desired application may be selected.

9. The method of claims **5**, **6**, **7** or **8** wherein providing the navigation procedures to a new application includes presenting a window at the display in which the user is presented with multiple, interactive command functions to effect navigation.

10. The method of claim **2** wherein the objects are stored at the respective reception systems in accordance with a predetermined plan, and wherein providing the first subgroup of command functions includes providing at a command for enabling the user to progress through a sequence of applications previously designated.

11. The method of claim **10** wherein the predetermined plan for storing objects at the respective reception systems includes providing the objects with a storage control parameter in their respective headers, and wherein providing the user with a command for progressing through a sequence of applications includes enabling the user to adjust the application sequence.

12. The method of claim **1** further including generating at least a third screen partition concurrently with the first and second screen partitions for presenting a second application and wherein the data structure of the objects includes a header and one or more data segments.

13. The method of claim **12** wherein the objects are stored at the respective reception systems in accordance with a predetermined plan, and wherein the predetermined plan for storing objects at the respective reception systems includes providing the objects with a storage control parameter in their respective headers, and wherein presenting a third screen partition includes presenting the second application as advertising.

14. The method of claim **1** further including generating one or more window partitions that overlays at least a portion of the application partition, the one or more windows for presenting data associated with the application displayed

and wherein the data structure of the objects includes a header and one or more data segments, and wherein the objects are stored at the respective reception systems in accordance with a predetermined plan, which includes providing the objects with a storage control parameter at their respective headers.

15. The method of claim **14** wherein generating window partitions includes providing the window partitions with fields for conducting interactive procedures associated with an underlying application.

16. The method of claim **15** wherein generating window partitions includes providing the window partitions with interactive fields for conducting transactional procedures.

17. The method of claim **1** wherein generating the first and second screen partitions includes generating the respective partitions at fixed, predetermined regions of the display screen, the second partition being arranged as a command bar and wherein the data structure of the objects includes a header and one or more data segments, and wherein the objects are stored at the respective reception systems in accordance with a predetermined plan, which includes providing the objects with a storage control parameter at their respective headers.

* * * * *

EXHIBIT B

US007072849B1

(12) **United States Patent**
Filepp et al.

(10) **Patent No.: US 7,072,849 B1**
(45) **Date of Patent: Jul. 4, 2006**

(54) **METHOD FOR PRESENTING ADVERTISING IN AN INTERACTIVE SERVICE**

(75) Inventors: **Robert Filepp**, White Plains, NY (US); **Alexander W. Bidwell**, New York, NY (US); **Francis C. Young**, Pearl River, NY (US); **Allan M. Wolf**, Ridgefield, CT (US); **Duane Tiemann**, Ossining, NY (US); **Mel Bellar**, New York, NY (US); **Robert D. Cohen**, Pouyhquag, NY (US); **James A. Galambos**, deceased, late of Westport, CT (US); **Kenneth H. Appleman**, Brewster, NY (US); **Sam Meo**, Carmel, NY (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/158,025**

(22) Filed: **Nov. 26, 1993**

Related U.S. Application Data

(60) Division of application No. 07/388,156, filed on Jul. 28, 1989, now Pat. No. 5,347,632, which is a continuation-in-part of application No. 07/328,790, filed on Mar. 23, 1989, now abandoned, which is a continuation-in-part of application No. 07/219,931, filed on Jul. 15, 1988, now abandoned.

(51) **Int. Cl.**
G06Q 30/00 (2006.01)

(52) **U.S. Cl.** **705/14**

(58) **Field of Classification Search** 364/401; 395/600, 144, 153, 200, 250, 201, 207, 210, 395/214, 611, 613, 614, 615, 762, 779, 782, 395/133, 135, 507, 327, 339, 340, 343, 346, 395/200.09, 445, 460
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,653,001 A * 3/1972 Ninke 395/132
(Continued)

FOREIGN PATENT DOCUMENTS

JP 573167 * 1/1982
JP 3204259 * 9/1991

OTHER PUBLICATIONS

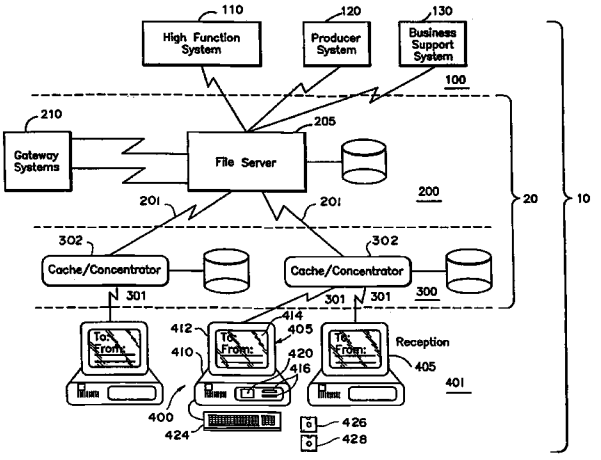
"Trintex Sets Prodigy Pricing; Telaction Reports New Cable System Affiliate"; *IDP Report*; v 9 Issue:n4 p. 2(2); Apr. 1, 1988; Dialog(file 648, 06639981).*
(Continued)

Primary Examiner—Donald L. Champagne
(74) *Attorney, Agent, or Firm*—Connolly Bove Lodge & Hutz LLP; Douglas Lefeve

(57) **ABSTRACT**

A method for presenting advertising in an interactive service provided on a computer network, the service featuring applications which include pre-created, interactive text/graphic sessions is described. The method features steps for presenting advertising concurrently with service applications at the user terminal configured as a reception system. In accordance with the method, the advertising is structured in a manner comparable to the service applications enabling the applications to be presented at a first portion of a display associated with the reception system and the advertising presented at a second portion. Further, steps are provided for storing and managing advertising at the user reception system so that advertising can be pre-fetched from the network and staged in anticipation of being called for presentation. This minimizes the potential for communication line interference between application and advertising traffic and makes the advertising available at the reception system so as not to delay presentation of the service applications. Yet further the method features steps for individualizing the advertising supplied to enhance potential user interest by providing advertising based on a characterization of the user as defined by the users interaction with the service, user demographics and geographical location. Yet additionally, advertising is provided with transactional facilities so that users can interact with it.

25 Claims, 16 Drawing Sheets



U.S. PATENT DOCUMENTS

4,552,349	A *	11/1985	Loos et al.	270/54
4,575,579	A *	3/1986	Simon et al.	178/4
4,688,167	A *	8/1987	Agarwal	395/343
4,714,996	A *	12/1987	Gladney et al.	395/600
4,805,134	A *	2/1989	Calo et al.	395/610
4,823,122	A *	4/1989	Mann et al.	340/825.28
4,873,662	A	8/1989	Sargent	
4,887,204	A *	12/1989	Johnson et al.	395/600
4,897,781	A *	1/1990	Chang et al.	395/600
4,897,782	A *	1/1990	Bennett et al.	395/600
4,989,850	A *	2/1991	Weller	270/1.1
5,036,314	A *	7/1991	Barillari et al.	340/717
5,087,805	A *	2/1992	Silverschotz et al.	219/121.71
5,105,184	A *	4/1992	Pirani et al.	340/721
5,119,290	A *	6/1992	Loo et al.	395/400

OTHER PUBLICATIONS

"The Handbook"; Prodigy; ©1990 Prodigy Services Company Glessbrenner, Alfred; Ceriés, New On-line fee; $4.95 a Month; *Home Office Computing*; v8 P. 36(1); Dec., 1990 Dialog (file 647, 09685321).*

"Advertisers Need Quick Fix for Zipping, Zapping"; *Marketing News*; v20 n10; pp. 12; May 9, 1986; Dialog: File 15, Acc# 00317906.*

"Consumers Plugging into new Electronic Mall"; *Advertising Age*; Mar. 4, 1985; p. 741; Dialog: File 16, Acc# 01155574.*

"CompuServe Will Jointly Offer Advertising and Direct Marketing Services via the CompuServe Information Service, a Videotex System"; *News Release*; Oct. 19, 1983; pp. 1-3; Dialog: File 16, Acc# 00962377.*

"Compuserve, L.M. Berry to Test Viability of Online Advertising"; *Online Database Report*; v4 n10; p. 12; Oct. 1983; Dialog: File 275, Acc# 00610155.*

Miller; "Database and Videotex Services—Where is Videotex Going?"; *Data Communications Buyers' Guide 1983*; pp. 152-157; Nov. 1982; Dialog: File 15, Acc# 00188062.*

Dictionary of Computers, Information Processing & Telecommunications, 2nd ed.; Jerry M. Rosenberg; 1984; pp. 183, 184, 268, 269, 303, 395, 402, 455, 530, 531, 594, 639, 640, 690, 691.*

Dictionary of Computers, Information Processing & Telecommunications, 2nd ed.; Jerry M Rosenberg; 1984; p. 700.*

Miller; "Database and Videotex Services—Where Is Videotex Going?"; *Data Communications Buyers' Guide 1983*; pp. 152, 157; Nov. 1982.*

Dietrich et al.; "Toward a Graphic Standard"; *PC World*; v2 n12; p. 264-269; Nov. 1984.*

"MCTel Inc. Advertises in the Electronic Mall Shop-at-Home Service, an Advertising Vehicle of CompuServe Inc. And L. M. Berry & Co."; *PR Newswire, PH303*; Jan. 23, 1985; Dialog: File 148, Acc# 02341095.*

"Consumers Plugging Into New Electronic Mall"; *Advertising Age*; Mar. 4, 1985; p. 741.*

"Home-Computer Shopping Arrives"; *Discount Store News*; v24; p. 3(2); Mar. 18, 1985; Dialog: File 148, Acc# 02324097.*

"Advertisers Need Quick Fix for Zipping, Zapping"; *Marketing News*; v20 n10; pp. 12; May 9, 1986.*

Caplinger, Michael, "An Information System Based on Distributed Objections", OOPSLA '87 Proceedings.

Schatz, Bruce, "Telesophy: A System for Manipulating the Knowledge of a Community", 1987 IEEE.

Christodoulakis, S., "The Multimedia Object Presentation Manager of MINOS: A Symmetric Approach", ACM SIGMOD Conf. 1986.

Christodoulakis, S., "Issues in the Architecture of a Document Archiver Using Optical Disk Technology", 1985 ACM.

Christodoulakis, S., "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and A System" 1986 ACM.

Sigel, Efrem, "The Future of Videotext", 1983, Knowledge Industry Publications, Inc., White Plains NY and London.

Alber, Antone F., *Videotex/Teletext Principles & Practices, McGraw-Hill, Inc.*, 1985.

* cited by examiner

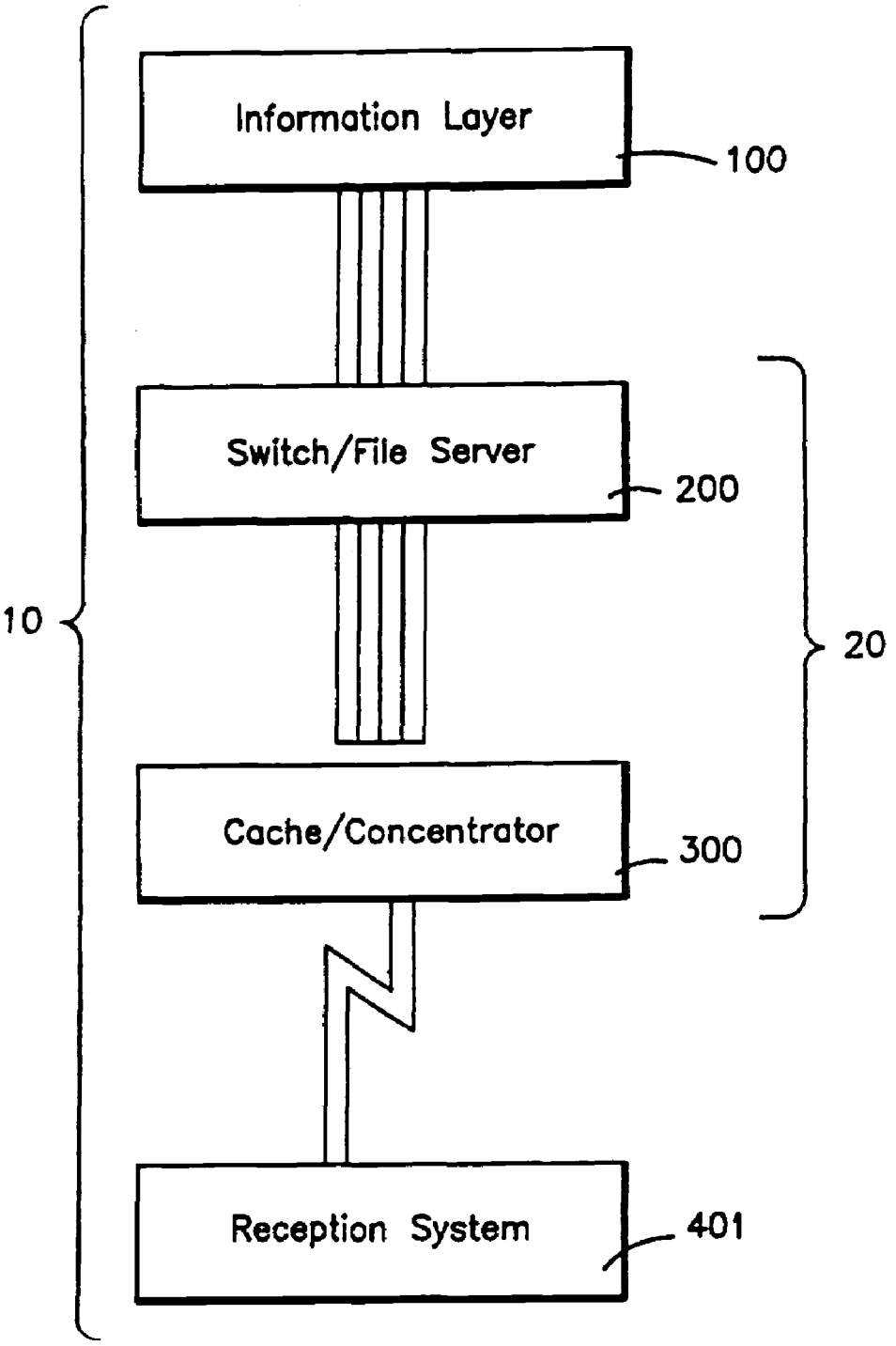

Information Layer
100
Switch/File Server
200
Cache/Concentrator
300
Reception System
401
10
20

FIG. 1

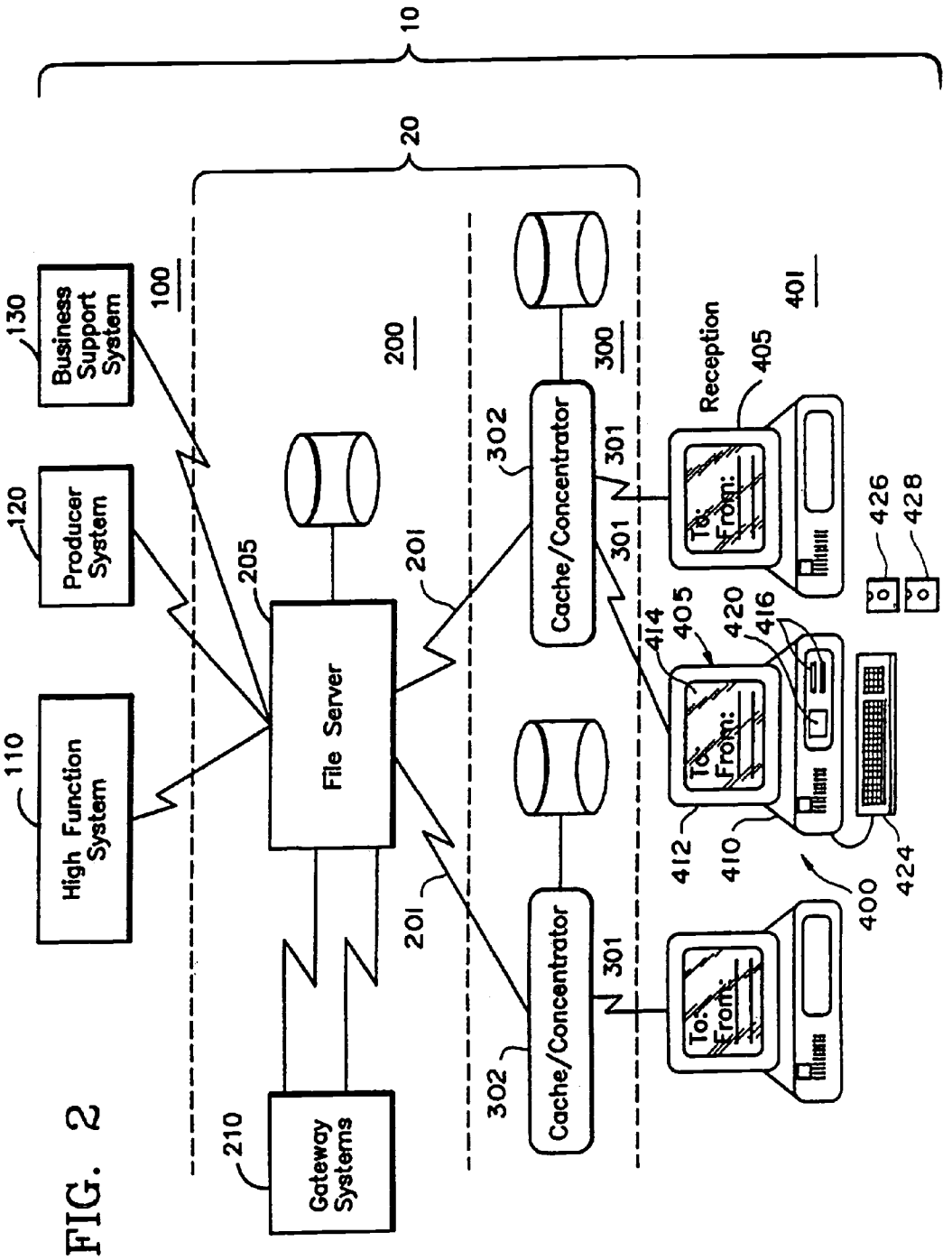

10
20
100
130
Business Support System
120
Producer System
110
High Function System
205
File Server
210
Gateway Systems
200
201
300
302
Cache/Concentrator
301
401
Reception
405
To:
From:
414
420
416
426
428
412
410
400
424

FIG. 2

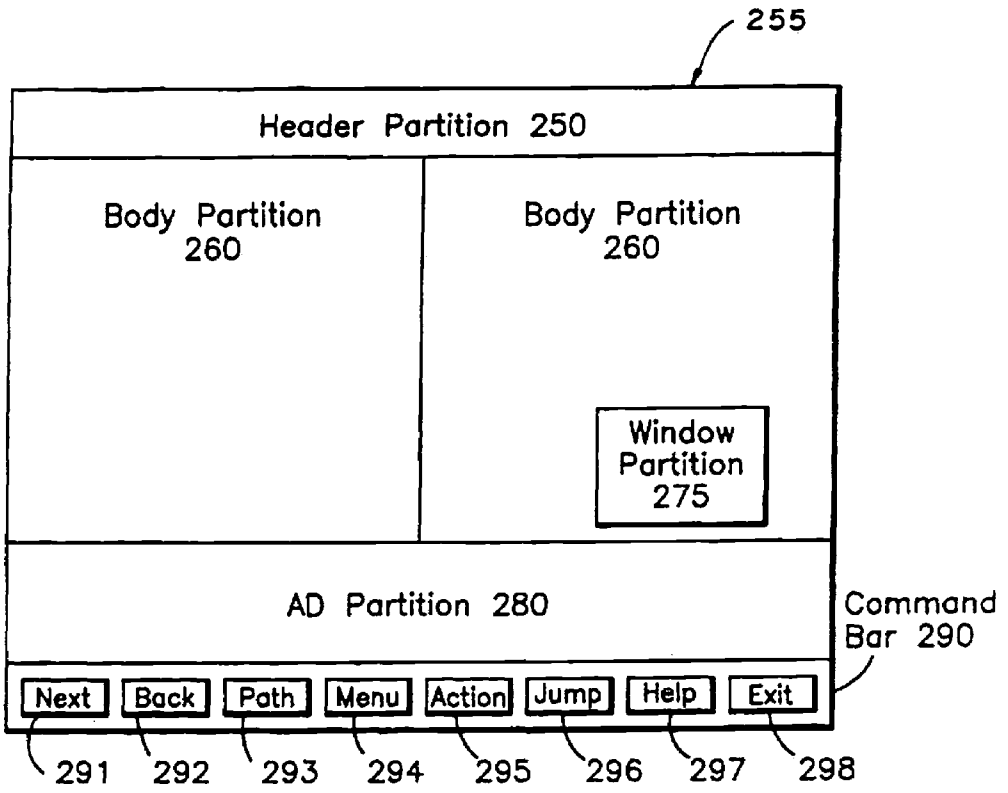

255
Header Partition 250
Body Partition 260
Body Partition 260
Window Partition 275
AD Partition 280
Command Bar 290
Next
Back
Path
Menu
Action
Jump
Help
Exit
291
292
293
294
295
296
297
298

FIG. 3a

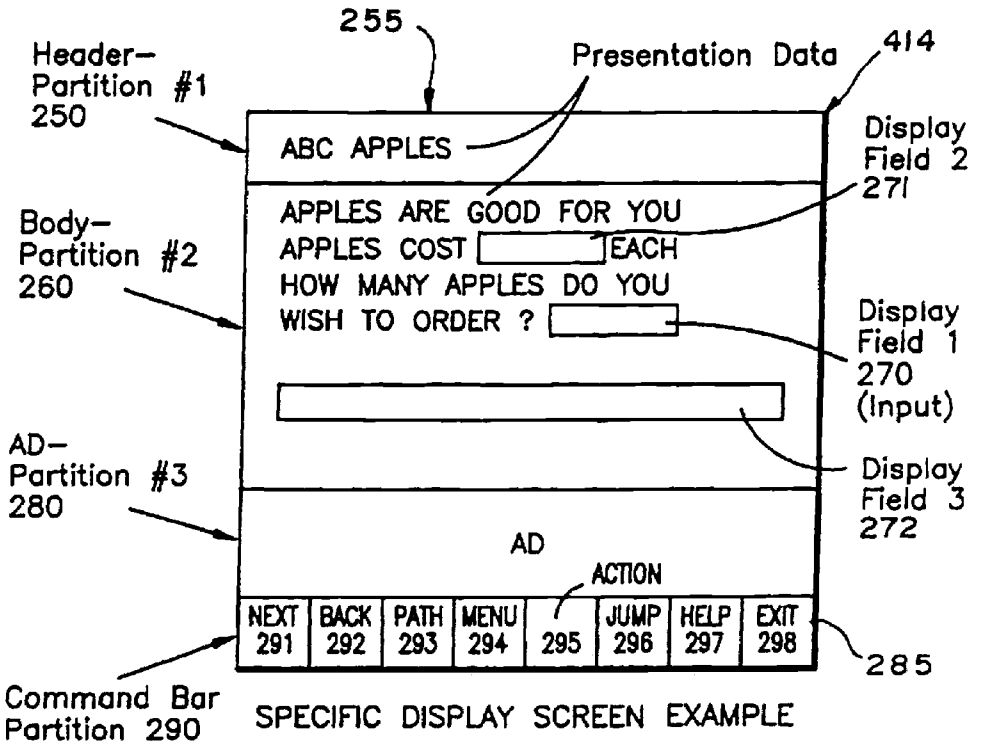

255
Header–
Partition #1
250
Presentation Data
414
ABC APPLES
Display
Field 2
271
APPLES ARE GOOD FOR YOU
APPLES COST EACH
HOW MANY APPLES DO YOU
WISH TO ORDER ?
Body–
Partition #2
260
Display
Field 1
270
(Input)
AD–
Partition #3
280
Display
Field 3
272
AD
ACTION
NEXT 291
BACK 292
PATH 293
MENU 294
295
JUMP 296
HELP 297
EXIT 298
285
Command Bar
Partition 290
SPECIFIC DISPLAY SCREEN EXAMPLE

FIG. 3b

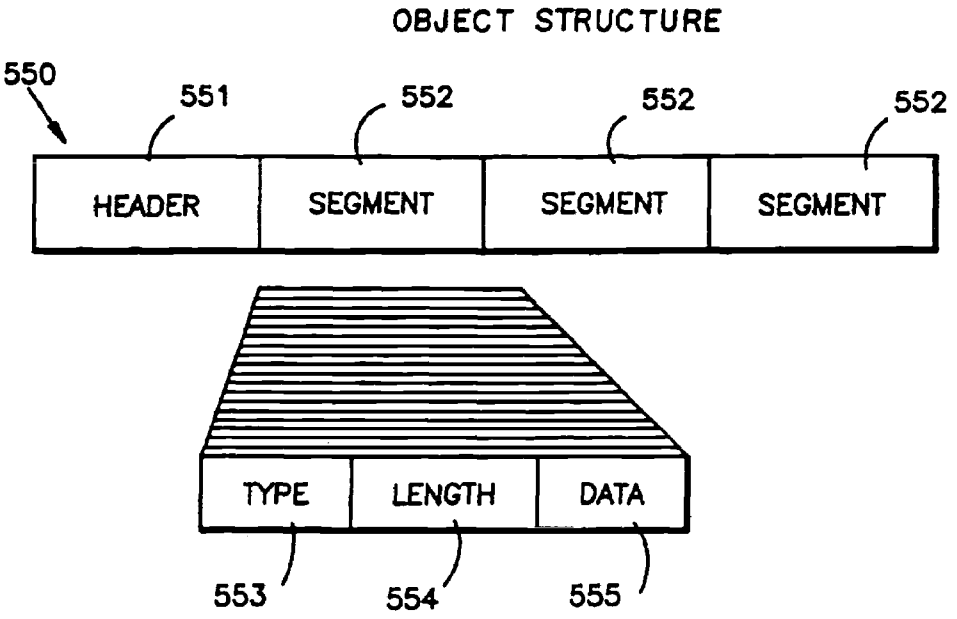

OBJECT STRUCTURE
550
551
552
552
552
HEADER
SEGMENT
SEGMENT
SEGMENT
TYPE
LENGTH
DATA
553
554
555

FIG. 4a

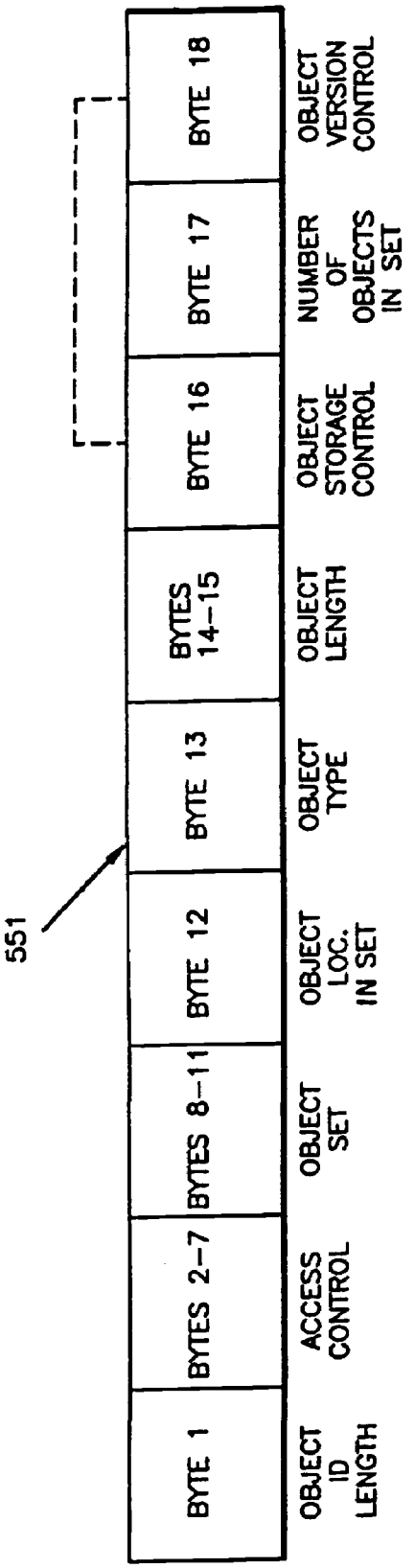

551
BYTE 1
BYTES 2–7
BYTES 8–11
BYTE 12
BYTE 13
BYTES 14–15
BYTE 16
BYTE 17
BYTE 18
OBJECT ID LENGTH
ACCESS CONTROL
OBJECT SET
OBJECT LOC. IN SET
OBJECT TYPE
OBJECT LENGTH
OBJECT STORAGE CONTROL
NUMBER OF OBJECTS IN SET
OBJECT VERSION CONTROL

FIG. 4b

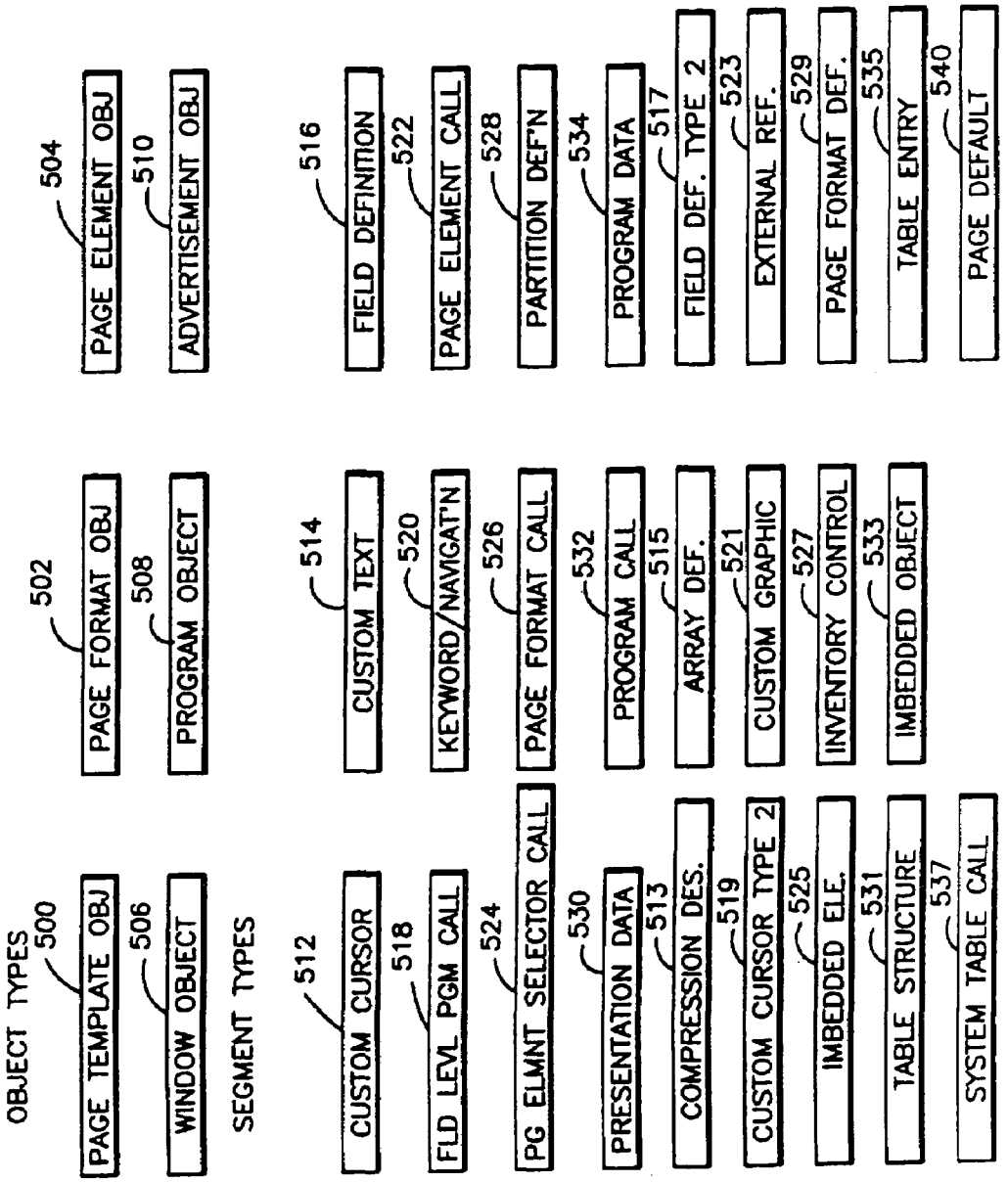

OBJECT TYPES
500
PAGE TEMPLATE OBJ
506
WINDOW OBJECT
502
PAGE FORMAT OBJ
508
PROGRAM OBJECT
504
PAGE ELEMENT OBJ
510
ADVERTISEMENT OBJ
SEGMENT TYPES
512
CUSTOM CURSOR
518
FLD LEVL PGM CALL
524
PG ELMNT SELECTOR CALL
530
PRESENTATION DATA
513
COMPRESSION DES.
519
CUSTOM CURSOR TYPE 2
525
IMBEDDED ELE.
531
TABLE STRUCTURE
537
SYSTEM TABLE CALL
514
CUSTOM TEXT
520
KEYWORD/NAVIGAT'N
526
PAGE FORMAT CALL
532
PROGRAM CALL
515
ARRAY DEF.
521
CUSTOM GRAPHIC
527
INVENTORY CONTROL
533
IMBEDDED OBJECT
516
FIELD DEFINITION
522
PAGE ELEMENT CALL
528
PARTITION DEF'N
534
PROGRAM DATA
517
FIELD DEF. TYPE 2
523
EXTERNAL REF.
529
PAGE FORMAT DEF.
535
TABLE ENTRY
540
PAGE DEFAULT

FIG. 4c

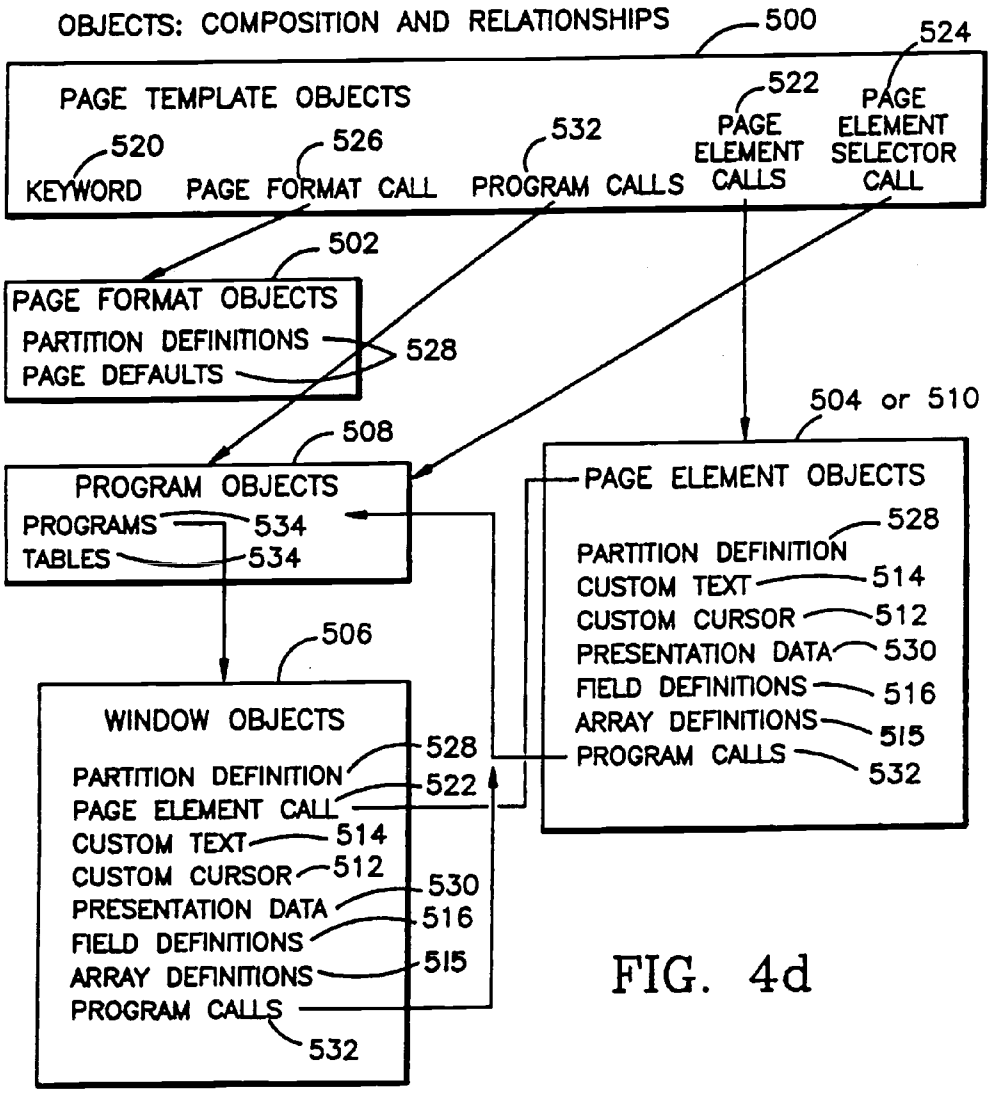

OBJECTS: COMPOSITION AND RELATIONSHIPS
500
PAGE TEMPLATE OBJECTS
524
522
520
526
532
KEYWORD
PAGE FORMAT CALL
PROGRAM CALLS
PAGE ELEMENT CALLS
PAGE ELEMENT SELECTOR CALL
502
PAGE FORMAT OBJECTS
PARTITION DEFINITIONS
PAGE DEFAULTS
528
504 or 510
508
PROGRAM OBJECTS
PROGRAMS
534
TABLES
534
PAGE ELEMENT OBJECTS
528
PARTITION DEFINITION
CUSTOM TEXT
514
CUSTOM CURSOR
512
PRESENTATION DATA
530
FIELD DEFINITIONS
516
ARRAY DEFINITIONS
515
PROGRAM CALLS
532
506
WINDOW OBJECTS
528
PARTITION DEFINITION
522
PAGE ELEMENT CALL
CUSTOM TEXT
514
CUSTOM CURSOR
512
530
PRESENTATION DATA
516
FIELD DEFINITIONS
515
ARRAY DEFINITIONS
PROGRAM CALLS
532

FIG. 4d

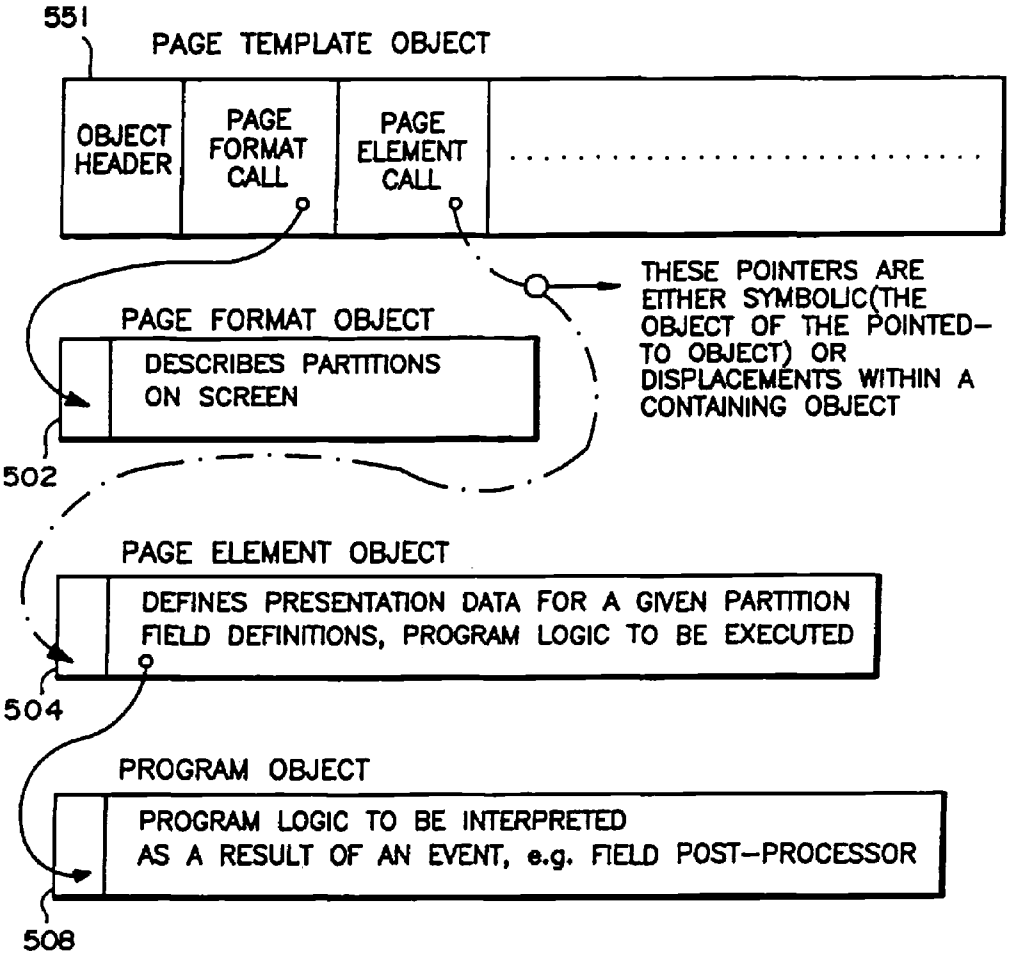

551
PAGE TEMPLATE OBJECT
OBJECT HEADER
PAGE FORMAT CALL
PAGE ELEMENT CALL
THESE POINTERS ARE EITHER SYMBOLIC(THE OBJECT OF THE POINTED-TO OBJECT) OR DISPLACEMENTS WITHIN A CONTAINING OBJECT
PAGE FORMAT OBJECT
DESCRIBES PARTITIONS ON SCREEN
502
PAGE ELEMENT OBJECT
DEFINES PRESENTATION DATA FOR A GIVEN PARTITION FIELD DEFINITIONS, PROGRAM LOGIC TO BE EXECUTED
504
PROGRAM OBJECT
PROGRAM LOGIC TO BE INTERPRETED AS A RESULT OF AN EVENT, e.g. FIELD POST-PROCESSOR
508

FIG. 5a

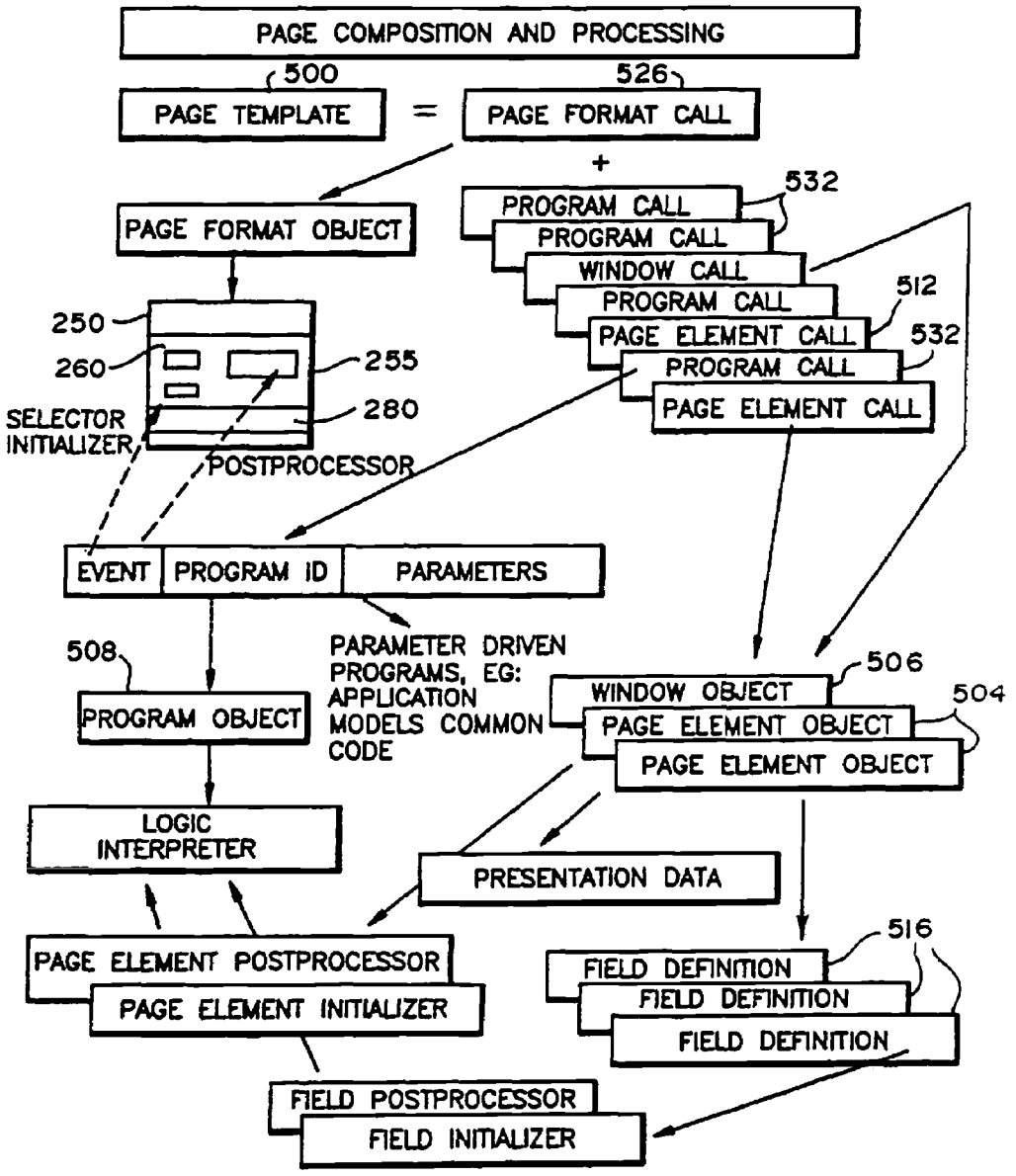
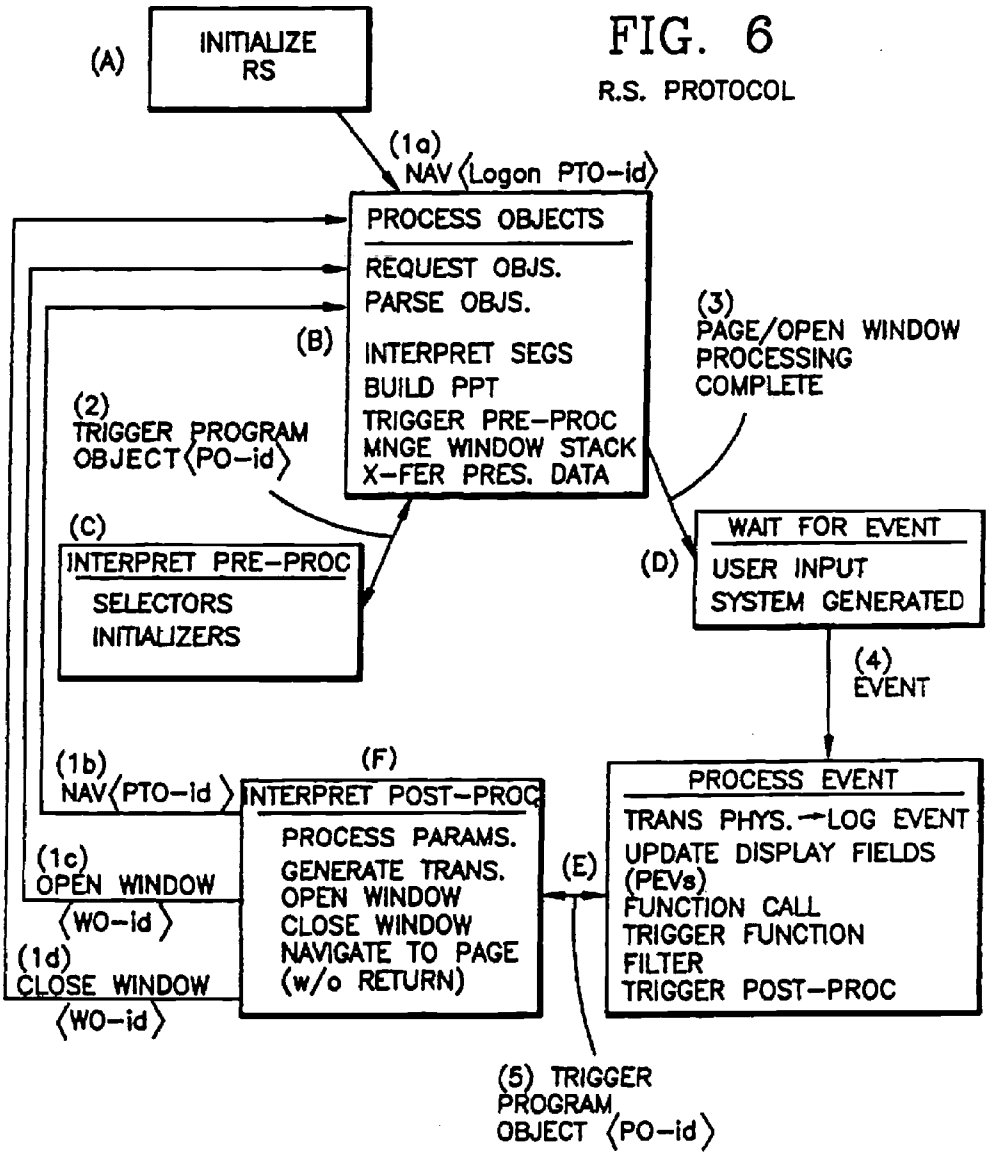
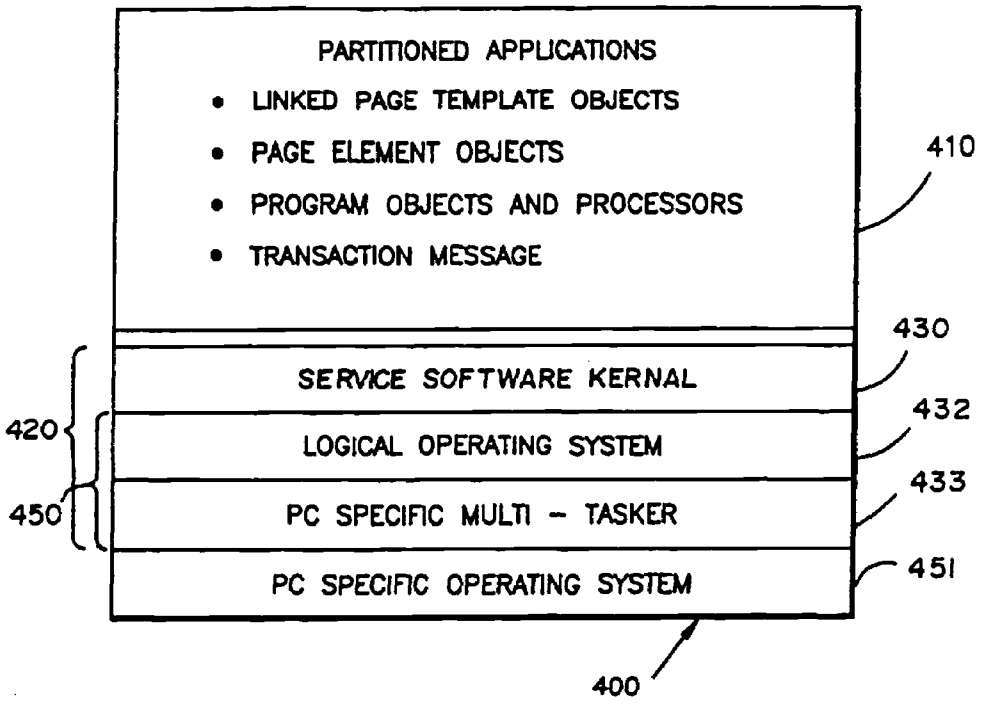

PAGE COMPOSITION AND PROCESSING
500
PAGE TEMPLATE
=
526
PAGE FORMAT CALL
+
PAGE FORMAT OBJECT
PROGRAM CALL
PROGRAM CALL
WINDOW CALL
PROGRAM CALL
PAGE ELEMENT CALL
PROGRAM CALL
PAGE ELEMENT CALL
532
512
532
250
260
255
280
SELECTOR INITIALIZER
POSTPROCESSOR
EVENT
PROGRAM ID
PARAMETERS
508
PROGRAM OBJECT
PARAMETER DRIVEN PROGRAMS, EG: APPLICATION MODELS COMMON CODE
506
WINDOW OBJECT
PAGE ELEMENT OBJECT
PAGE ELEMENT OBJECT
504
LOGIC INTERPRETER
PRESENTATION DATA
PAGE ELEMENT POSTPROCESSOR
PAGE ELEMENT INITIALIZER
516
FIELD DEFINITION
FIELD DEFINITION
FIELD DEFINITION
FIELD POSTPROCESSOR
FIELD INITIALIZER

FIG. 5b

FIG. 6

R.S. PROTOCOL





RECEPTION SYSTEM LAYERS

FIG. 7

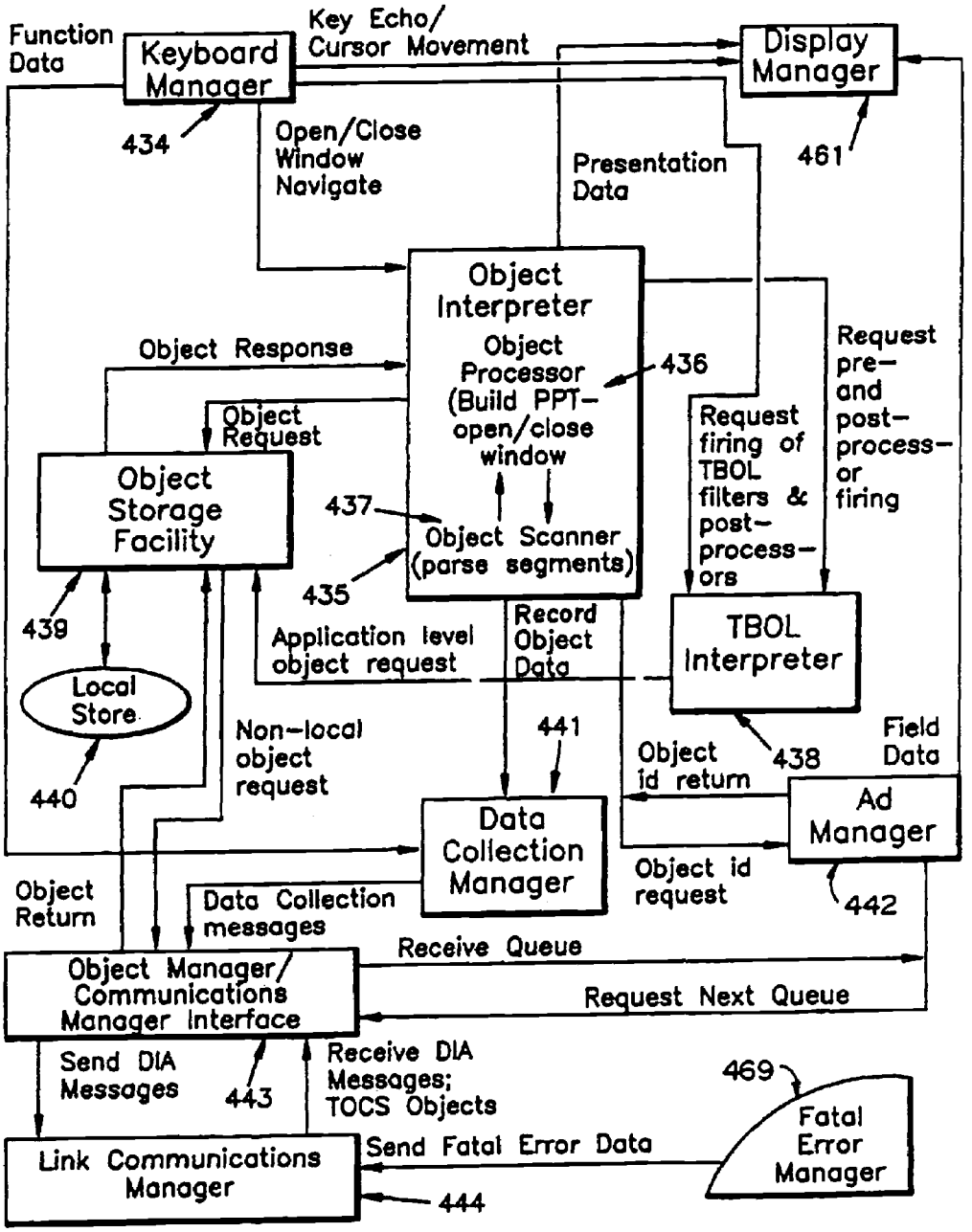

Function Data
Keyboard Manager
Key Echo/ Cursor Movement
Display Manager
434
461
Open/Close Window Navigate
Presentation Data
Object Interpreter
Object Processor (Build PPT- open/close window
436
Object Response
Object Request
Request pre- and post- process- or firing
Request firing of TBOL filters & post- process- ors
Object Storage Facility
437
Object Scanner (parse segments)
435
439
Application level object request
Record Object Data
TBOL Interpreter
Local Store
440
Non-local object request
441
Object id return
438
Field Data
Ad Manager
Data Collection Manager
Object id request
442
Object Return
Data Collection messages
Receive Queue
Object Manager/ Communications Manager Interface
Request Next Queue
Send DIA Messages
443
Receive DIA Messages; TOCS Objects
469
Fatal Error Manager
Send Fatal Error Data
Link Communications Manager
444

FIG. 8

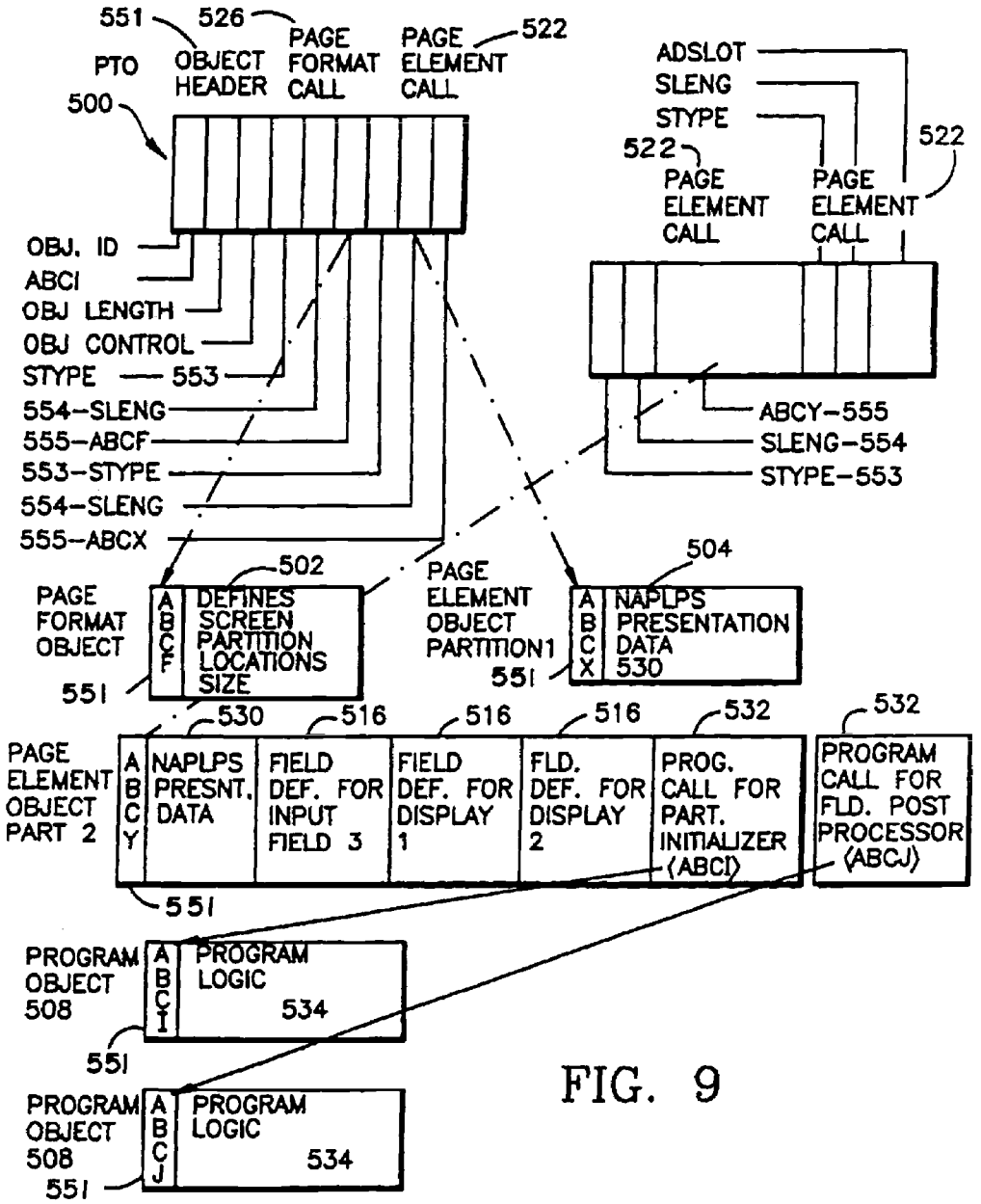

551
526
PAGE FORMAT CALL
PAGE ELEMENT CALL
522
PTO
OBJECT HEADER
500
OBJ. ID
ABCI
OBJ LENGTH
OBJ CONTROL
STYPE
553
554-SLENG
555-ABCF
553-STYPE
554-SLENG
555-ABCX
ADSLOT
SLENG
STYPE
522
PAGE ELEMENT CALL
PAGE ELEMENT CALL
522
ABCY-555
SLENG-554
STYPE-553
502
PAGE FORMAT OBJECT
A B C F
DEFINES SCREEN PARTITION LOCATIONS SIZE
551
PAGE ELEMENT OBJECT PARTITION1
504
A B C X
NAPLPS PRESENTATION DATA
530
551
530
516
516
516
532
532
PAGE ELEMENT OBJECT PART 2
A B C Y
NAPLPS PRESNT. DATA
FIELD DEF. FOR INPUT FIELD 3
FIELD DEF. FOR DISPLAY 1
FLD. DEF. FOR DISPLAY 2
PROG. CALL FOR PART. INITIALIZER (ABCI)
PROGRAM CALL FOR FLD. POST PROCESSOR (ABCJ)
551
PROGRAM OBJECT 508
A B C I
PROGRAM LOGIC
534
551
PROGRAM OBJECT 508
A B C J
PROGRAM LOGIC
534
551

FIG. 9

FIG. 10

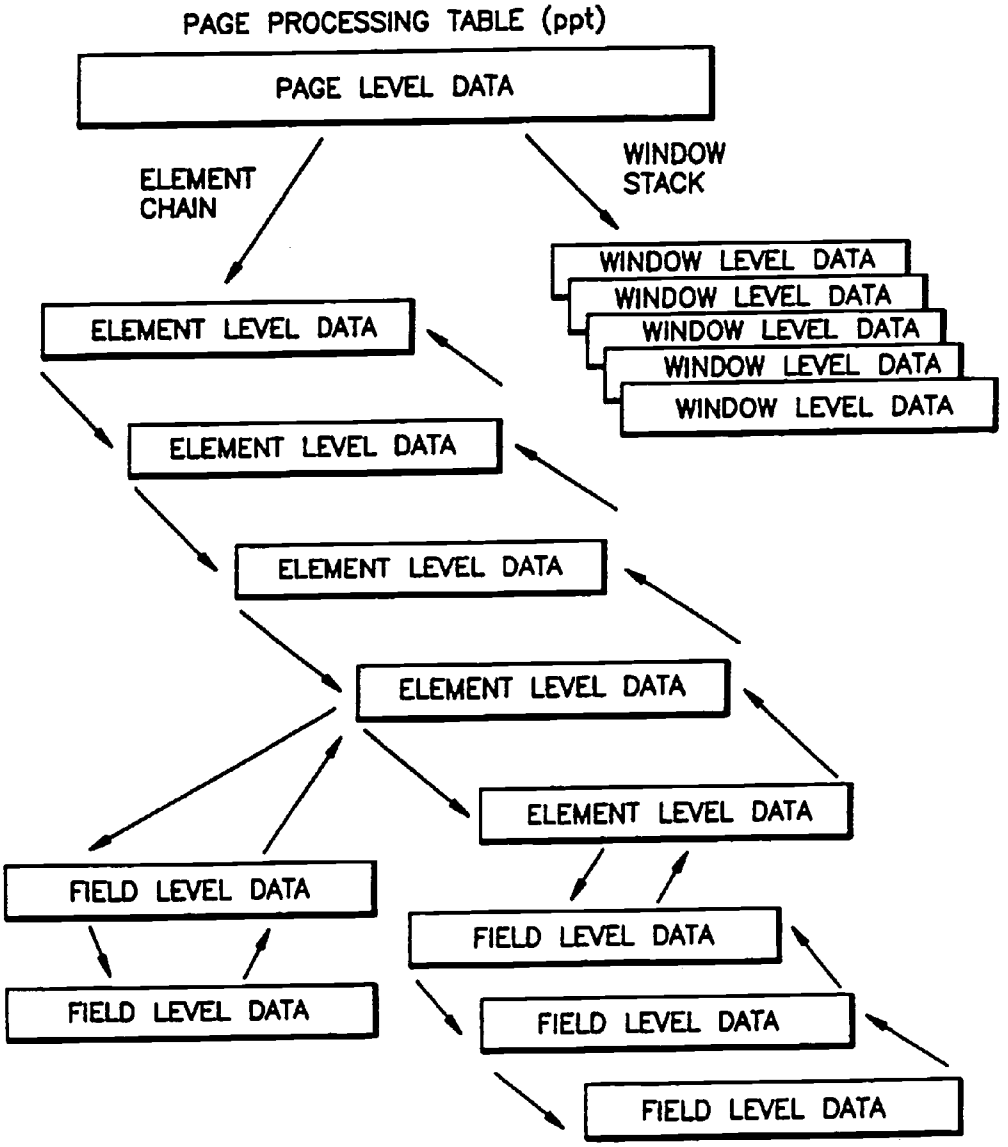
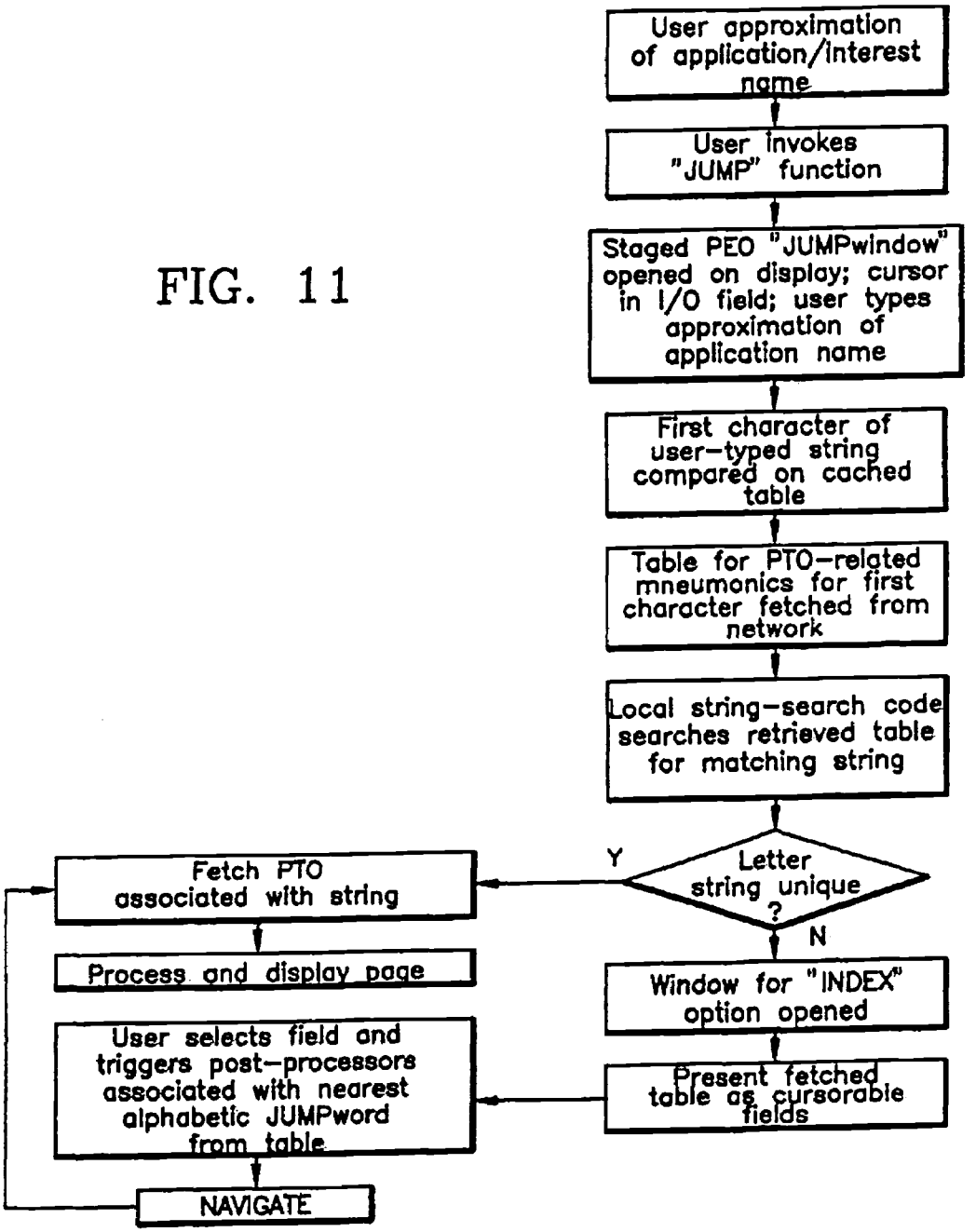

PAGE PROCESSING TABLE (ppt)
PAGE LEVEL DATA
ELEMENT CHAIN
WINDOW STACK
WINDOW LEVEL DATA
WINDOW LEVEL DATA
WINDOW LEVEL DATA
WINDOW LEVEL DATA
WINDOW LEVEL DATA
ELEMENT LEVEL DATA
ELEMENT LEVEL DATA
ELEMENT LEVEL DATA
ELEMENT LEVEL DATA
ELEMENT LEVEL DATA
FIELD LEVEL DATA
FIELD LEVEL DATA
FIELD LEVEL DATA
FIELD LEVEL DATA
FIELD LEVEL DATA

FIG. 11


User approximation of application/interest name
User invokes "JUMP" function
Staged PEO "JUMPwindow" opened on display; cursor in I/O field; user types approximation of application name
First character of user-typed string compared on cached table
Table for PTO-related mneumonics for first character fetched from network
Local string-search code searches retrieved table for matching string
Letter string unique ?
Y
N
Fetch PTO associated with string
Process and display page
User selects field and triggers post-processors associated with nearest alphabetic JUMPword from table
NAVIGATE
Window for "INDEX" option opened
Present fetched table as cursorable fields

METHOD FOR PRESENTING ADVERTISING IN AN INTERACTIVE SERVICE

RELATED APPLICATIONS

This is a division of application Ser. No. 07/388,156 filed Jul. 28, 1989, Sep. 13, 1994, as U.S. Pat. No. 5,347,632, application Ser. No. 07/388,156 being a continuation in part of application Ser. No. 07/328,790, now abandoned filed Mar. 23, 1989, which itself was a continuation in part of application Ser. No. 07/219,931, now abandoned filed Jul. 15, 1988.

BACKGROUND OF THE INVENTION

1. Field of Use

This invention relates generally to a distributed processing, interactive computer network intended to provide very large numbers of simultaneous users; e.g. millions, access to an interactive service having large numbers; e.g., thousands, of applications which include pre-created, interactive text/graphic sessions; and more particularly, to a method for presenting advertising to service users during interactive sessions, the method featuring steps for presenting advertising concurrently with applications, the advertising being organized as data which is stored for presentation and replenished at the user sites so as to minimize interference with retrieval and presentation of application data; the method also featuring steps for individualizing the advertising presented based on user characterizations defined by service interaction and/or other data such as user demographics and geographical location.

2. Prior Art

Interactive computer networks are not new. Traditionally they have included conventional, hierarchical architectures wherein a central, host computer responds to the information requests of multiple users. An illustration would be a time-sharing network in which multiple users, each at a remote terminal, log onto a host that provides data and software resource for sequentially receiving user data processing requests, executing them and supplying responses back to the users.

While such networks have been successful in making the processing power of large computers available to many users, problems have existed with them. For example, in such networks, the host has been required to satisfy all the user data processing requests. As a result, processing bottlenecks arise at the host that cause network slowdowns and compel expansion in computing resources; i.e., bigger and more complex computer facilities, where response times are sought to be held low in the face of increasing user populations.

Host size and complexity, however, are liabilities for interactive networks recently introduced to offer large numbers of the public access to transactional services such as home shopping, banking, and investment maintenance, as well as informational services concerning entertainment, business and personal matters. As can be appreciated, commercial interactive networks will have to provide attractive services at low cost and with minimal response times in order to be successful. Unlike military and governmental networks where, because of the compulsory nature of the service performed, costs, content and efficiency are of secondary concern, in commercial services, since use is predominantly elective, and paid for by the consumer, costs will have to be held low, content made interesting and response times reduced in order to attract and hold both users who would subscribe to the service and merchandisers who would rely on it as a channel of distribution for their good and services. Accordingly, if the service delivery system is allowed to increase in size and complexity, either unchecked or unsubsidized, higher use costs would have to be charged to recover the larger capital and operating expenses, with the negatively, spiralling effect that fewer users could be attracted and be available over which to spread the costs for sustaining the service.

In the past, other suppliers of mass-media services such as radio, television, newspapers, and magazines, have sought to hold access and subscription prices to affordable levels by relying on advertising income to offset the costs of providing their users with the benefits of technological advance. However, in the case of interactive computer services, it has not been apparent how advertising could be introduced without adversely affecting service speed and content quality, which as noted, are considered essential elements for service success.

Particularly, in an interactive service, if advertising were provided in a conventional manner; as for example, by providing the advertising as additional data to be supplied to and presented at the user sites, the effort would compete with the supplying and presentation of service application data, and have the undesirable effect of diminishing service response time. More specifically, if advertising were supplied conventionally from a host to a user site, the application traffic, which constitutes the substance of the service, would have to compete with advertising for network communication resources. Yet additionally, even if traffic conflicts were somehow avoided, the presentation of the service applications would have to be interrupted and delayed; for example like television and radio commercials, as advertising content was presented to the user. The effect of these anticipated delays would be to degrade application response time and diminishing service attractiveness.

Additionally, in view of the need to maintain the user's interest in application content so as to drive the interactive session, it has not been apparent how advertising matter could be provided without distracting the user or disrupting the session. Where service response time is diminished for the sake of advertising which is either irrelevant or distasteful, insult is added to the injury, increasing the likelihood the user, and service, will be, simply, turned off.

SUMMARY OF INVENTION

Accordingly, it is an object of this invention to provide a method for presenting advertising in an interactive service.

It is another object of this invention to provide a method for presenting advertising in an interactive service which method enables the presentation of advertising to be integrated with presentation of service applications.

It is a yet another object of this invention to provide a method for presenting advertising in an interactive service which method minimizes the potential for interference between the supply of interactive-service applications and advertising.

It is a still another object of this invention to provide a method for presenting advertising which minimizes the potential for interference between presentation of interactive-service applications and advertising. It is yet a further object of this invention to provide a method for presenting advertising in an interactive service which method enables the advertising presented to be individualized to the user to whom it is presented in order to increase the likelihood the advertising will be of interest to the user.

And, it is still a further object of this invention to provide a method for presenting advertising in an interactive service which method enables the user to transactionally interact with the advertising presented.

Briefly, the method for presenting advertising in accordance with this invention achieves the above-noted and other objects by featuring steps for presenting advertising concurrently with service applications at the user reception system; i.e., terminal. In accordance with the method, the advertising is structured in a manner comparable to the manner in which the service applications are structured. This enables the applications to be presented at a first portion of a display associated with the reception system and the advertising to be presented concurrently at a second portion of the display. Further, in accordance with the method, the user reception system at which the advertising is presented includes facility for storing and managing the advertising so that it can be pre-fetched from the network and staged at the reception system in anticipation of being called for presentation. This minimizes the potential for communication line interference between application and advertising traffic and makes the advertising available at the reception system so as not to delay presentation of the service applications. Yet further the method features steps for individualizing the advertising supplied to enhance potential user interest by providing advertising based on a characterization of the user as defined by the users interaction with the service, user demographics and geographical location. Yet additionally, advertising is provided with transactional facilities so that users can interact with it.

In preferred form, the method includes step for organizing advertising and applications as objects that collectively include presentation data and executable program instructions for generating the advertising and applications at the reception system. In accordance with the preferred form of the method, advertising and application objects are selectively distributed in the service network in accordance with a predetermined plan based on the likelihood the applications and advertising will be called by the respective user reception systems.

Also in preferred form, the method includes step for maintaining an advertising object identification queue, and an advertising object store that are replenished based on predetermined criteria as advertising is called for association and presentation with applications. In accordance with the method, as applications are executed at the reception system, the application objects provide generalized calls for advertising. The application calls for advertising are subsequently forwarded to the reception system advertising queue management facility which, in turn supplies an identification of advertising who's selection has been individualized to the user based on, as noted, the user's prior interaction history with the service, demographics and local. Thereafter, the object identification for the advertising is passed to the object store to determine if the object is available at the reception system. In preferred form, if the advertising object is not available at the reception system, a sequence of alternative advertising object identifications can be provided which if also are unavailable at the reception system will resulting in an advertising object being requested from the network. In this way, advertising of interest can be targeted to the user and secured in time-efficient manner to increase the likelihood of user interest and avoid service distraction.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further objects, features and advantages of the invention will become clear from the following more detailed description when read with reference to the accompanying drawings in which:

FIG. **1** is a block diagram of the interactive computer network in which the method of the present invention may be practiced;

FIG. **2** is a schematic diagram of the network illustrated in FIG. **1**;

FIGS. **3***a* and **3***b* are plan views of a display screen for a user reception system at which advertising can be presented to a user in accordance with the method of the present invention;

FIGS. **4***a*, **4***b*, **4***c* and **4***d* are schematic drawings that illustrate the structure of objects, and object segments that may be used for advertising and applications in accordance with the method of the present invention;

FIG. **5***a* is a schematic diagram that illustrates the configuration of the page template object which might be used for presentation of an application and advertising in accordance with the method of the present invention;

FIG. **5***b* is a schematic diagram that illustrates page composition which might be used for presentation of an application and advertising in accordance with the method of the present invention;

FIG. **6** is a schematic diagram that illustrates the protocol which might be used by a reception system for supporting applications and advertising in accordance with the method of the present invention;

FIG. **7** is a schematic diagram that illustrates major layers for a reception system which might be used for supporting applications and advertising in accordance with the method of the present invention;

FIG. **8** is a block diagram that illustrates native code modules for a reception system which might be used for supporting applications and advertising in accordance with the method of the present invention;

FIG. **9** is a schematic diagram that illustrates an example of a partitioned application to be processed by a reception system which might be used for supporting applications and advertising in accordance with the method of the present invention;

FIG. **10** illustrates generation of a page with a page processing table for a reception system which might be used for supporting applications and advertising in accordance with the method of the present invention;

FIG. **11** is a flow diagram for an aspect of the navigation method of a reception system which might be used for supporting applications and advertising in accordance with the method of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

General System Description

FIGS. **1** and **2** show a network in which the method of the present invention for presenting advertising might be used. As seen the network, designated **10**, includes a plurality of reception units within a reception layer **401** for displaying information and providing transactional services. In this arrangement, many users each access network **10** with a conventional personal computer; e.g., one of the IBM or IBM-compatible type, which has been provided with application software to constitute a reception system (RS) **400**.

As seen in FIG. **1**, interactive network **10** uses a layered structure that includes an information layer **100**, a switch/file server layer **200**, and cache/concentrator layer **300** as well as reception layer **401**. This structure maintains active application databases and delivers requested parts of the databases on demand to the plurality of RS **400**'s, shown in FIG. **2**. As seen in FIG. **2**, cache/concentrator layer **300** includes a plurality of cache/concentrator units **302**, each or which serve a plurality of RS **400** units over lines **301**. Additionally, switch/file server layer **200** is seen to include a server unit **205** connected to multiple cache/concentrator units **302** over lines **201**. Still further, server unit **205** is seen to be connected to information layer **100** and its various elements, which act as means for producing, supplying and maintaining the network databases and other information necessary to support network **10**. Continuing, switch/filer layer **200** is also seen to include gateway systems **210** connected to server **205**. Gateways **210** couple layer **200** to other sources of information and data; e.g., other computer systems. As will be appreciated by those skilled in the art, layer **200**, like layers **401** and **300**, could also include multiple servers, gateways and information layers in the event even larger numbers of users were sought to be served.

Continuing with reference to FIG. **2**, in preferred form, each RS **400** is seen to include a personal computer **405** having a CPU **410** including a microprocessor (as for example, one of the types made by INTEL Corporation in its X'86 family of microprocessors), companion RAM and ROM memory and other associated elements, such as monitor **412** with screen **414** and a keyboard **424**. Further, personal computer **405** may also include one or two floppy disk drives **416** for receiving diskettes **426** containing application software used to support the interactive service and facilitate the interactive sessions with network **10**. Additionally, personal computer **405** would include operating systems software; e.g., MS-DOS, supplied on diskettes **428** suitable for the personal computer being used. Personal computer **405** still further may also include a hard-disk drive **420** for storing the application software and operating system software which may be transferred from diskettes **426** and **428** respectfully.

Once so configured, each RS **400** provides: a common interface to other elements of interactive computer network **10**; a common environment for application processing; and a common protocol for user-application conversation which is independent of the personal computer brand used. RS **400** thus constitutes a universal terminal for which only one version of all applications on network **10** need be prepared, thereby rendering the applications interpretable by a variety of brands of personal computers.

RS **400** formulated in this fashion is capable of communication with the host system to receive information containing either of two types of data, namely objects and messages. Objects have a uniform, self-defining format known to RS **400**, and include data types, such as interpretable programs and presentation data for display at monitor screen **414** of the user's personal computer **405**. Applications presented at RS **400** are partitioned into objects which represent the minimal units available from the higher levels of interactive network **10** or RS **400**. In this arrangement, each application partition typically represents one screen or a partial screen of information, including fields filled with data used in transactions with network **10**. Each such screen, commonly called a page, is represented by its parts and is described in a page template object, discussed below.

Applications, having been partitioned into minimal units, are available from higher elements of network **10** or RS **400**, and are retrieved on demand by RS **400** for interpretive execution. Thus, not all partitions of a partitioned application need be resident at RS **400** to process a selected partition, thereby raising the storage efficiency of the user's RS **400** and minimizing response time. Each application partition is an independent, self-contained unit and can operate correctly by itself. Each partition may refer to other partitions either statically or dynamically. Static references are built into the partitioned application, while dynamic references are created from the execution of program logic using a set of parameters, such as user demographics or locale. Partitions may be chosen as part of the RS processing in response to user created events, or by selecting a key word of the partitioned application (e.g., "JUMP" or "INDEX," discussed below), which provides random access to all services represented by partitioned applications having key words.

Objects provide a means of packaging and distributing partitioned applications. As noted, objects make up one or more partitioned applications, and are retrieved on demand by a user's RS **400** for interpretive execution and selective storage. All objects are interpreted by RS **400**, thereby enabling applications to be developed independently of the personal computer brand used.

Objects may be nested within one another or referenced by an object identifier (object-id) from within their data structure. References to objects permit the size of objects to be minimized. Further, the time required to display a page is minimized when referenced objects are stored locally at RS **400** (which storage is determined by prior usage meeting certain retention criteria), or have been pre-fetched, or in fact, are already used for the current page.

Objects carry application program instructions and/or information for display at monitor screen **414** of RS **400**. Application program objects, called pre-processors and post-processors, set up the environment for the user's interaction with network **10** and respond to events created when the user inputs information at keyboard **424** of RS **400**. Such events typically trigger a program object to be processed, causing one of the following: sending of transactional information to the coapplications in one layer of the network **10**; the receiving of information for use in programs or for presentation in application-dependent fields on monitor screen **414**; or the requesting of a new objects to be processed by RS **400**. Such objects may be part of the same application or a completely new application.

The RS **400** supports a protocol by which the user and the partitioned applications communicate. All partitioned applications are designed knowing that this protocol will be supported in RS **400**. Hence, replication of the protocol in each partitioned application is avoided, thereby minimizing the size of the partitioned application.

RS **400** includes a means to communicate with network **10** to retrieve objects in response to events occurring at RS **400** and to send and receive messages.

RS **400** includes a means to selectively store objects according to a predetermined storage criterion, thus enabling frequently used objects to be stored locally at the RS, and causing infrequently used objects to forfeit their local storage location. The currency of objects stored locally at the RS **400** is verified before use according to the object's storage control parameters and the storage criterion in use for version checking.

Selective storage tailors the contents of the RS **400** memory to contain objects representing all or significant parts of partitioned applications favored by the user.

Because selective storage of objects is local, response time is reduced for those partitioned applications that the user accesses most frequently.

Since much of the application processing formerly done by a host computer in previously known time-sharing networks is now performed at the user's RS **400**, the higher elements of network **10**, particularly layer **200**, have as their primary functions the routing of messages, serving of objects, and line concentration. The narrowed functional load of the higher network elements permits many more users to be serviced within the same bounds of computer power and I/O capability of conventional host-centered architectures.

Network **10** provides information on a wide variety of topics, including, but not limited to news, industry, financial needs, hobbies and cultural interests. Network **10** thus eliminates the need to consult multiple information sources, giving users an efficient and timesaving overview of subjects that interest them.

The transactional features of interactive network **10** saves the user time, money, and frustration by reducing time spent traveling, standing in line, and communicating with sales personnel. The user may, through RS **400**, bank, send and receive messages, review advertising provided in accordance with the method of the present invention, place orders for merchandise, and perform other transactions.

In preferred form, network **10** provides information, advertising and transaction processing services for a large number of users simultaneously accessing the network via the public switched telephone network (PSTN), broadcast, and/or other media with their RS **400** units. Services available to the user include display of information such as movie reviews, the latest news, airlines reservations, the purchase of items such as retail merchandise and groceries, and quotes and buy/sell orders for stocks and bonds. Network **10** provides an environment in which a user, via RS **400** establishes a session with the network and accesses a large number of services. These services are specifically constructed applications which as noted are partitioned so they may be distributed without undue transmission time, and may be processed and selectively stored on a user's RS **400** unit.

System Configuration

As shown in FIG. **1**, interactive computer network **10** includes four layers: information layer **100**, switch and file server layer **200**, concentrator layer **300**, and reception layer **401**.

Information layer **100** handles: (1) the production, storage and dissemination of data and (2) the collection and off-line processing of such data from each RS session with the network **10** so as to permit the targeting of information and advertising to be presented to users and for traditional business support.

Switch and file server layer **200** and cache/concentrator layer **300** together constitute a delivery system **20** which delivers requested data to the RS **400**'s of reception layer **401** and routes data entered by the user or collected at RS **400**'s to the proper application in network **10**. With reference to FIG. **2**, the information used in a RS **400** either resides locally at the RS **400**, or is available on demand from the cache/concentrator **300** or the file server **205**, via the gateway **210**, which may be coupled to external providers, or is available from information layer **100**.

There are two types of information in the network **10** which are utilized by the RS **400**: objects and messages.

Objects include the information requested and utilized by the RS **400** to permit a user to select specific parts of applications, control the flow of information relating to the applications, and to supply information to the network. Objects are self-describing structures organized in accordance with a specific data object architecture, described below. Objects are used to package presentation data and program instructions required to support the partitioned applications and advertising presented at a RS **400**. Objects are distributed on demand throughout interactive network **10**. Objects may contain: control information; program instructions to set up an application processing environment and to process user or network created events; information about what is to be displayed and how it is to be displayed; references to programs to be interpretively executed; and references to other objects, which may be called based upon certain conditions or the occurrence of certain events at the user's personal computer, resulting in the selection and retrieval of other partitioned applications packaged as objects.

Messages are information provided by the user or the network and are used in fields defined within the constructs of an object, and are seen on the user's RS monitor **412**, or are used for data processing at RS **400**. Additionally, and as more fully described hereafter, messages are the primary means for communication within and without the network. The format of messages is application dependent. If the message is input by the user, it is formatted by the partitioned application currently being processed on RS **400**. Likewise, and with reference to FIG. **2**, if the data are provided from a co-application database residing in delivery system **20**, or accessed via gateway **210** or high function system **110** within the information layer **100**, the partitioned application currently being processed on RS **400** causes the message data to be displayed in fields on the user's display monitor as defined by the particular partitioned application.

All active objects reside in file server **205**. Inactive objects or objects in preparation reside in producer system **120**. Objects recently introduced into delivery system **20** from the producer system **120** will be available from file server **205**, but, may not be available on cache/concentrator **302** to which the user's RS **400** has dialed. If such objects are requested by the RS **400**, the cache/concentrator **302** automatically requests the object from file server **205**. The requested object is routed back to the requesting cache/concentrator **302**, which automatically routes it to the communications line on which the request was originally made, from which it is received by the RS **400**.

The RS **400** is the point of application session control because it has the ability to select and randomly access objects representing all or part of partitioned applications and their data. RS **400** processes objects according to information contained therein and events created by the user on personal computer **405**.

Applications on network **10** act in concert with the distributed partitioned applications running on RS **400**. Partitioned applications constructed as groups of objects and are distributed on demand to a user's RS **400**. An application partition represents the minimum amount of information and program logic needed to present a page or window, i.e. portion of a page presented to the user, perform transactions with the interactive network **10**, and perform traditional data processing operations, as required, including selecting another partitioned application to be processed upon a user generated completion event for the current partitioned application.

Objects representing all or part of partitioned applications may be stored in a user's RS **400** if the objects meet certain criteria, such as being non-volatile, non-critical to network integrity, or if they are critical to ensuring reasonable response time. Such objects are either provided on diskettes **426** together with RS **400** system software used during the installation procedure or they are automatically requested by RS **400** when the user makes selections requiring objects not present in RS **400**. In the latter case, RS **400** requests from cache/concentrator layer **300** only the objects necessary to execute the desired partitioned application.

Reception system application software **426** in preferred form is provided for IBM and IBM-compatible brands of personal computers **405**, and all partitioned applications are constructed according to a single architecture which each such RS **400** supports. With reference to FIG. **2**, to access network **10**, a user preferably has a personal computer **405** with at least 512K RAM and a single disk drive **416**. The user typically accesses network **10** using a 1,200 or 2,400 bps modem (not shown). To initiate a session with network **10**, objects representing the logon application are retrieved from the user's personal diskette, including the R.S. application software, which was previously set up during standard installation and enrollment procedures with network **10**. Once communication between RS **400** and cache/concentrator layer **300** has been established, the user begins a standard logon procedure by inputting a personal entry code. Once the logon procedure is complete, the user can begin to access various desired services (i.e., partitioned applications) which provide display of requested information and/or transaction operations.

Applications and Pages

Applications, i.e. information events, are composed of a sequence of one or more pages opened at screen **414** of monitor **412**. This is better seen with reference to FIGS. **3***a* and **3***b* were a page **255** is illustrated as might appear at screen **414** of monitor **412**. With reference to FIG. **3***a*, each page **255** is formatted with a service interface having page partitions **250**, **260**, **280**, and **290** (not to be confused with application partitions). Window page partitions **275**, well known in the art, are also available and are opened and closed conditionally on page **255** upon the occurrence of an event specified in the application being run. Each page partition **250**, **260**, **280** and **290** and window **275** is made up of a page element which defines the content of the partition or window.

Each page **255** includes: a header page partition **250**, which has a page element associated with it and which typically conveys information on the page's topic or sponsor; one or more body page partitions **260** and window page partitions **275**, each of which is associated with a page element which as noted gives the informational and transactional content of the page. For example, a page element may contain presentation data selected as a menu option in the previous page, and/or may contain prompts to which a user responds in pre-defined fields to execute transactions. As illustrated in FIG. **3***b*, the page element associated with body page partition **260**—includes display fields **270**, **271**, **272**. A window page partition **275** seen in FIG. **3***a* represents the same informational and transactional capability as a body partition, except greater flexibility is provided for its location and size.

Continuing with reference to FIG. **3***a*, in accordance with the invention, advertising **280** is provided over network **10**, like page elements, also includes information for display on page **255**, and may be included in any partition of a page. Advertising **280** is presented to the user on an individualized basis from queues of advertising object identifications (ids) that are constructed off-line by business system **130**, and sent to file server **205** where they are accessible to each RS **400**.

Individualized queues of advertising object ids are constructed based upon data collected on the partitioned applications that were accessed by a user, and upon events the user generated in response to applications. The data are collected and reported by RS **400** to a data collection co-application in file server **205** for later transmission to business system **130**. In addition to application access and use characteristics, a variety of other parameters, such as user demographics or postal ZIP code, may be used as targeting criteria. From such data, queues of advertising object ids are constructed that are targeted to either individual users or to sets of users who fall into certain groups according to such parameters. Stated otherwise, the advertising presented is individualized to the respective users based on characterizations of the respective users as defined by the interaction history with the service and such other information as user demographics and locale. As will be appreciated by those skilled in the art, conventional marketing analysis techniques can be employed to establish the user characterizations based on the collected application usage data above noted and other information.

Also with reference to FIG. **3***b*, the service interface is seen to include a command region **285** which enables the user to interact with the network RS **400** and other elements of network **10**, so as to cause such operations as navigating from page to page, performing a transaction, or obtaining more information about other applications. As shown in FIG. **3***b*, interface region **285** includes a command bar **290**—having a number of commands **291**–**298** which the user can execute. The functions of commands **291**–**298** are discussed in greater detail below.

Network Objects

As noted above, in conventional time-sharing computer networks, the data and program instructions necessary to support user sessions are maintained at a central host computer. However, that approach has been found to create processing bottlenecks as greater numbers of users are connected to the network; bottlenecks which require increases in processing power and complexity; e.g., multiple hosts of greater computing capability, if the network is to meet demand. Further, such bottlenecks have been found to also slow response time as more users are connected to the network and seek to have their requests for data processing answered.

The consequences of the host processing bottlenecking is to either compel capital expenditures to expand host processing capability, or accept longer response times; i.e., a slower network, and risk user dissatisfaction.

However, even in the case where additional computing power is added, and where response time is allowed to increase, eventually the host becomes user saturated as more and more users are sought to be served by the network. The network described above, however, is designed to alleviate the effects of host-centered limitations, and extend the network saturation point. This objective is achieved by reducing the demand on the host for processing resources by structuring the network so that the higher network levels act primarily to maintain and supply data and programs to the

lower levels of the network, particularly RS **400**, which acts to manage and sustain the user screen displays.

More particularly, the described network features procedures for parsing the network data and program instructions required to support the interactive user sessions into packets, referred to as objects, and distributing them into the network where they can be processed at lower levels, particularly, reception system **400**.

In accordance with the method of the present invention, the screens presented at the user's monitor are each divided into addressable partitions shown in FIG. **3***a*, and the display text and graphics necessary to make up the partitions, as well as the program instructions and control data necessary to deliver and sustain the screens and partitions, are formulated from pre-created objects. Further, the objects are structured in accordance with an architecture that permits the displayed data to be relocatable on the screen, and to be reusable to make up other screens and other sessions, either as pre-created and stored sessions or interactive sessions, dynamically created in response to the user's requests.

As shown in FIG. **4***c*, the network objects are organized as a family of objects each of which perform a specific function in support of the interactive session. More particularly, in accordance with the preferred form of the invention, the network object family is seen to include 6 members: page format objects **502**, page element objects **504**, window objects **506**, program objects **508**, advertisement objects **510** and page template objects **500**.

Within this family, page format objects **502** are designed to define the partitioning **250** to **290** of the monitor screen shown in FIG. **3***a*. The page format objects **502** provide a means for pre-defining screen partitions and for ensuring a uniform look to the page presented on the reception system monitor. They provide the origin; i.e., drawing points, and dimensions of each page partition and different values for presentation commands such as palette and background color.

Page format objects **502** are referenced whenever non-window data is to be displayed and as noted ensure a consistent presentation of the page. In addition, page format objects **502** assures proper tessellation or "tiling" of the displayed partitions.

Page element objects **504**, on the other hand, are structured to contain the display data; i.e., text and graphic, to be displayed which is mapped within screen partitions **250** to **290**, and to further provide the associated control data and programs. More specifically, the display data is described within the object as NAPLPS data, and includes, PDI, ASCII, Incremental Point and other display encoding schemes. Page element objects also control the functionality within the screen partition by means of field definition segments **516** and program call segments **532**, as further described in connection with the description of such segments hereafter. Page element objects **504** are relocatable and may be reused by many pages. To enable the displayable data to be relocated, display data must be created by producers in the NAPLPS relative mode.

Continuing with reference to FIG. **4***c*, window objects **506** include the display and control data necessary to support window partitions **275** best seen in FIG. **3***a*. Windows contain display data which overlay the base page and control data which supersede the base page control data for the underlying screen during the duration of the window. Window objects **506** contain data which is to be displayed or otherwise presented to the viewer which is relatively independent from the rest of the page. Display data within windows overlay the base page until the window is closed. Logic associated with the window supersedes base page logic for the duration of the window. When a window is opened, the bit map of the area covered by window is saved and most logic functions for the overlaid page are deactivated. When the window is closed, the saved bit map is swapped onto the screen, the logic functions associated with the window are disabled, and prior logic functions are reactivated.

Windows are opened by user or program control. They do not form part of the base page. Windows would typically be opened as a result of the completion of events specified in program call segments **532**.

Window objects **506** are very similar in structure to page element objects **504**. The critical difference is that window objects **506** specify their own size and absolute screen location by means of a partition definition segment **528**.

Program objects **508** contain program instructions written in a high-level language called TRINTEX Basic Object Language, i.e., TBOL, described in greater detail hereafter, which may be executed on RS **400** to support the application. More particularly, program objects **508** include interpretable program code, executable machine code and parameters to be acted upon in conjunction with the presentation of text and graphics to the reception system monitors.

Program objects **508** may be called for execution by means of program call segments **532**, which specify when a program is to be executed (event), what program to execute (program pointer), and how programs should run (parameters).

Programs are treated as objects to conform to the open-ended design philosophy of the data object architecture (DOA), allowing the dissemination of newly developed programs to be easily and economically performed. As noted above, it is desirable to have as many of these program objects staged for execution at or as close to RS **400** as possible.

Still further, in accordance with the method of the present invention, advertising objects **510** include the text and graphics that may be presented at ad partition **280** presented on the monitor screen as shown in FIG. **3***b*.

Finally, the object family includes page template objects **500**. Page template objects **500** are designed to define the components of the full screen presented to the viewer. Particularly, page template objects **500** include the entry point to a screen, the name of the page format objects which specify the various partitions a screen will have and the page element object that contain the display data and partitioning parameters for the page.

Additionally, page template object **500** includes the specific program calls required to execute the screens associated with the application being presented to the user, and may serve as the means for the user to selectively move through; i.e., navigate the pages of interest which are associated with various applications. Thus, in effect, page template objects **500** constitute the "recipe" for making up the collection of text and graphic information required to make the screens to be presented to the user.

Also in accordance with the invention, object **500** to **510** shown in FIG. **4***c* are themselves made up of further sub-blocks of information that may be selectively collected to define the objects and resulting pages that ultimately constitute the application presented to the user in an interactive text and graphic session.

More specifically and as shown schematically in FIG. **4***a*, objects **500** to **510** are predefined, variable length records consisting of a fixed length header **551** and one or more

self-defining record segments **552** a list of which is presented in FIG. **4***c* as segment types **512** to **540**.

In accordance with this design, and as shown in FIG. **4***b*, object header **551** in preferred form is 18 bytes in length and contains a prescribed sequence of information which provides data regarding the object's identification, its anticipated use, association to other objects, its length and its version and currency.

More particularly, each of the 18 bytes of object header **551** are conventional hexadecimal, 8 bit bytes and are arranged in a fixed pattern to facilitate interpretation by network **10**. Particularly, and as shown in FIG. **4***b*, the first byte of header **551**; i.e., byte **1**, identifies the length of the object ID in hexadecimal. The next six bytes; i.e., bytes **2** to **7**, are allocated for identifying access control to the object so as to allow creation of closed user groups to whom the object(s) is to be provided. As will be appreciated by those skilled in the art, the ability to earmark objects in anticipation of user requests enables the network anticipate requests and pre-collect objects from large numbers of them maintained to render the network more efficient and reduce response time. The following 4 bytes of header **551**; bytes **8** to **11**, are used to identify the set of objects to which the subject object belongs. In this regard, it will be appreciated that, again, for speed of access and efficiency of selection, the objects are arranged in groups or sets which are likely to be presented to user sequentially in presenting the page sets; i.e., screens that go to make up a session.

Following identification of the object set, the next byte in header **551**; i.e., byte **12**, gives the location of the subject object in the set. As will be appreciated here also the identification is provided to facilitate ease of object location and access among the many thousands of objects that are maintained to, thereby, render their selection and presentation more efficient and speedy.

Thereafter, the following bytes of header **551**; i.e., byte **13**, designates the object type; e.g., page format, page template, page element, etc. Following identification of the object type, two bytes; i.e., bytes **14**, **15**, are allocated to define the length of the object, which may be of whatever length is necessary to supply the data necessary, and thereby provides great flexibility for creation of the screens. Thereafter, a single byte; i.e., byte **16**, is allocated to identify the storage characteristic for the object; i.e., the criterion which establishes at what level in network **10** the object will be stored, and the basis upon which it will be updated. At least a portion of this byte; i.e, the higher order nibble (first 4 bits reading from left to right) is associated with the last byte; i.e., byte **18**, in the header which identifies the version of the object, a control used in determining how often in a predetermined period of time the object will be updated by the network.

Following storage characteristic byte **16**, header **551** includes a byte; i.e., **17**, which identifies the number of objects in the set to which the subject object belongs. Finally, and as noted above, header **551** includes a byte; i.e., **18**, which identifies the version of the object. Particularly the object version is a number to establish the control for the update of the object that are resident at RS **400**.

As shown in FIG. **4***a*, and as noted above, in addition to header **551**, the object includes one more of the various segment types shown in FIG. **4***c*.

Segments **512** to **540** are the basic building blocks of the objects. And, as in the case of the object, the segments are also self-defining. As will be appreciated by those skilled in the art, by making the segments self-defining, changes in the objects and their use in the network can be made without changing pre-existing objects.

As in the case of objects, the segments have also been provided with a specific structure. Particularly, and as shown in FIG. **4***a*, segments **552** consists of a designation of segment type **553**, identification of segment length **554**, followed by the information necessary to implement the segment and its associated object **555**; e.g., either, control data, display data or program code.

In this structure, segment type **553** is identified with a one-byte hexadecimal code which describes the general function of the segment. Thereafter, segment length **554** is identified as a fixed two-byte long field which carries the segment length as a hexadecimal number in INTEL format; i.e., least significant byte first. Finally, data within segments may be identified either by position or keyword, depending on the specific requirements of the segment.

The specific structure for the objects and segments in shown in FIG. **4***c* and is described below. In that description the following notation convention is used:

< >—mandatory item

( )—optional item

. . . —item may be repeated

|item| |item|

< > ( )—items in a column indicate either/or

|item| |item|

The structure for objects is:

PAGE TEMPLATE OBJECT,

[<header> (compression descriptor) <page format call> (page element call) . . . (program call) . . . (page element selector) (system table call) . . . external reference) (keyword/navigation) . . .];

As noted above, page format objects **502** are designed to define the partitioning **250** to **290** of monitor screen **414** shown in FIG. **3***a*.

PAGE FORMAT OBJECT,

[<header> (compression descriptor) (page defaults) <partition definition>];

PAGE ELEMENT OBJECT,

[<header> (compression descriptor) (presentation data) . . . (program call) . . . (custom cursor) . . . (custom text) . . . (field definition) . . . (field-level program call) . . . (custom cursor type 2) . . . (custom graphic) . . . (field definition type 2) . . . (array definition) . . . (inventory control)];

Page element objects, as explained, are structured to contain the display data; i.e., text and graphics, to be presented at screen partitions **250** to **290**.

WINDOW OBJECT,

[<header> (compression description) <partition definition> (page element call) (presentation data) . . . (program call) . . . (custom cursor) . . . (custom text) . . . (custom cursor type 2) . . . (custom graphic) . . . (field definition) . . . (field level program call) . . . (field definition type 2) . . . (array definition) . . . (inventory control)];

As noted, window objects include display and control data necessary to support window partition at screen **414**.

PROGRAM OBJECTS,

[<header> (compression descriptor) <program data> . . .].

Program objects, on the other hand, contain program instructions written in higher-level language which may be executed at RS **400** to support the application.

ADVERTISEMENT OBJECT,

[<header> (compression descriptor) (presentation data) . . . (program call) . . . (custom cursor) . . . (custom text) . . . (field definition) . . . (field-level program call) . . . (custom cursor type 2) . . . (custom graphic) . . . (field definition type 2) . . . (array definition) . . . (inventory control)];

In accordance with the invention, and as can be seen, advertisement objects are substantially the same as page element objects, with the difference being that, as their name implies, their subject matter is selected to concern advertising.

Continuing, the structure for the object segments follows from the above description, and is as described more fully in parent application Ser. No. 388,156 now issued as U.S. Pat. No. 5,347,632, the contents of which patent are incorporated herein by reference.

Network Messages

In addition to the network objects, and the display data, control data, and the program instructions they contain as previously described, network **10** also exchanges information regarding the support of user sessions and the maintenance of the network as "messenger". Specifically, messages typically relate to the exchange of information associated with initial logon of a reception system **400** to network **10**, dialogue between RS **400** and other elements and communications by the other network elements amongst themselves.

To facilitate message exchange internally, and through gateway **210** to entities externally to network **10**, a protocol termed the "Data Interchange Architecture" (DIA) is used to support the transport and interpretation of information. More particularly, DIA enables: communications between RS **400** units, separation of functions between network layers **100**, **200**, **300** and **401**; consistent parsing of data; an "open" architecture for network **10**; downward compatibility within the network; compatibility with standard industry protocols such as the IBM System Network Architecture; Open Systems Interconnections standard; support of network utility sessions; and standardization of common network and application return codes.

Thus DIA binds the various components of network **10** into a coherent entity by providing a common data stream for communications management purposes. DIA provides the ability to route messages between applications based in IBM System Network Architecture (SNA), (well known in the art, and more fully described in *Data and Computer Communications*, by W. Stallings, Chapter 12, McMillian Publishing, Inc. (1985)) and non-SNA reception system applications; e.g. home computer applications. Further, DIA provides common data structure between applications run at RS **400** units and applications that may be run on external computer networks; e.g. Dow Jones Services, accessed through gateway **210**. As well, DIA provides support for utility sessions between backbone applications run within network **10**. A more detailed description of network messaging in provided in above noted U.S. Pat. No. 5,347,632, the content of which is incorporated herein by reference.

Object Language

In accordance with the design of network **10**, in order to enable the manipulation of the network objects, the application programs necessary to support the interactive text/graphic sessions are written in a high-level language referred to as "TBOL", (TRINTEX Basic Object Language, "TRINTEX" being the former company name of one of the assignees of this invention). TBOL is specifically adapted for writing the application programs so that the programs may be compiled into a compact data stream that can be interpreted by the application software operating in the user personal computer, the application software being designed to establish the network Reception System **400** previously noted and described in more detail hereafter.

The Reception System application software supports an interactive text/graphics sessions by managing objects. As explained above, objects specify the format and provide the content; i.e., the text and graphics, displayed on the user's screen so as to make up the pages that constitute the application. As also explained, pages are divided into separate areas called "partitions" by certain objects, while certain other objects describe windows which can be opened on the pages. Further, still other objects contain TBOL application programs which facilitate the data processing necessary to present the pages and their associated text and graphics.

As noted, the object architecture allows logical events to be specified in the object definitions. An example of a logical event is the completion of data entry on a screen; i.e., an application page. Logical events are mapped to physical events such as the user pressing the <ENTER> key on the keyboard. Other logical events might be the initial display of a screen page or the completion of data entry in a field. Logical events specified in page and window object definitions can be associated with the call of TBOL program objects.

RS **400** is aware of the occurrence of all physical events during the interactive text/graphic sessions. When a physical event such as depression of the forward <TAB> key corresponds to a logical event such as completion of data entry in a field, the appropriate TBOL program is executed if specified in the object definition. Accordingly, the TBOL programs can be thought of as routines which are given control to perform initialization and post-processing application logic associated with the fields, partitions and screens at the text/graphic sessions.

RS **400** run time environment uses the TBOL programs and their high-level key-word commands called verbs to provide all the system services needed to support a text/graphic session, particularly, display management, user input, local and remote data access.

TBOL programs have a structure that includes three sections: a header section in which the program name is specified; a data section in which the data structure the program will use are defined; and a code section in which the program logic is provided composed of one or more procedures. More specifically, the code section procedures are composed of procedure statements, each of which begins with a TBOL key word called a verb.

The name of a procedure can also be used as the verb in a procedure statement exactly as if it were a TBOL key-word verb. This feature enables a programmer to extend the language vocabulary to include customized application-oriented verb commands.

Continuing, TBOL programs have a program syntax that includes a series of "identifiers" which are the names and labels assigned to programs, procedures, and data structures.

An identifier may be up to 31 characters long; contain only uppercase or lowercase letters A through Z, digits 0 through 9, and/or the special character underscore (); and must begin with a letter. Included among the system identifiers are: "header section identifiers" used in the header section for the program name; "data section identifiers" used

in the data section for data structure names, field names and array names; and finally, "code section identifiers" used in the code section for identification of procedure names and statement labels. A more detailed description of TBOL is provided in parent application Ser. No. 388,156 now issued as U.S. Pat. No. 5,347,632, the contents of which patent are incorporated herein by reference.

Reception System Operation

RS **400** of computer system network **10** uses software called native code modules (described below) to enable the user to select options and functions presented on the monitor screen **414** of personal computer **405**, to execute partitioned applications and to process user created events, enabling the partitioned application to interact with network **10**. Through this interaction, the user is able to input data into fields provided as part of the display, or may individually select choices causing a standard or personalized page to be built (as explained below) for display on the monitor of personal computer **405**. Such inputs will cause RS **400** to interpret events and trigger pre-processors or post-processors, retrieve specified objects, communicate with system components, control user options, cause the display of advertisements on a page, open or close window partitions to provide additional navigation possibilities, and collect and report data about events, including certain types of objects processed. For example, the user may select a particular option, such as opening or closing window partition **275**, which is present on the monitor and follow the selection with a completion key stroke, such as ENTER. When the completion keystroke is made, the selection is translated into a logical event that triggers the execution of a post-processor, (i.e., a partitioned application program object) to process the contents of the field.

Functions supporting the user-partitioned application interface can be performed using the command bar **290**, or its equivalent using pull down windows or an overlapping cascade of windows. These functions can be implemented as part of the RS native functions or can be treated as another partition(s) defined for every page for which an appropriate set of supporting objects exist and remain resident at RS **400**. If the functions are part of RS **400**, they can be altered or extended by verbs defined in the RS virtual machine that permit the execution of program objects to be triggered when certain functions are called, providing maximum flexibility.

To explain the functions the use of a command bar is assumed. Command bar **290** is shown in FIGS. **3***a* and **3***b* and includes a NEXT command **291**, a BACK command **292**, a PATH command **293**, a MENU command **294**, an ACTION command **295**, a JUMP command **296**, a HELP command **297**, and an EXIT command **298**.

NEXT command **291** causes the next page in the current page set to be built. If the last page of a page set has already been reached, NEXT command **291** is disabled by RS **400**, avoiding the presentation of an invalid option.

BACK command **292** causes the previous page of the current page set to be built. If the present page is the first in the page set, BACK command **292** is disabled, since it is not a valid option.

A filter program can be attached to both the NEXT or BACK functions to modify their implicit sequential nature based upon the value of the occurrence in the object set id.

PATH command **293** causes the next page to be built and displayed from a list of pages that the user has entered, starting from the first entry for every new session.

MENU command **294** causes the page presenting the previous set of choices to be rebuilt.

ACTION command **295** initiates an application dependent operation such as causing a new application partition to be interpreted, a window partition **275** to be opened and enables the user to input any information required which may result in a transaction or selection of another window or page.

JUMP command **296** causes window partition **275** to be opened, allowing the user to input a keyword or to specify one from an index that may be selected for display.

HELP command **297** causes a new application partition to be interpreted such as a HELP window pertaining to where the cursor is positioned to be displayed in order to assist the user regarding the present page, a particular partition, or a field in a page element.

EXIT command **298** causes a LOGOFF page template object (PTO) to be built, and a page logoff sequence to be presented at RS **400** monitor screen **414**.

Navigation Interface

Continuing, as a further feature, network **10** includes an improved procedure for searching and retrieving applications from the store of applications distributed throughout network **10**; e.g., server **205**, cache/concentrator **302** and RS **400**. More specifically, the procedure features use of precreated search tables which represent subsets of the information on the network arranged with reference to the page template objects (PTO) and object-ids of the available applications so that in accordance with the procedure, the relevant tables and associated objects can be provided to and searched at the requesting RS **400** without need to search the entire store of applications on the network. As will be appreciated, this reduces the demand on the server **205** for locating and retrieving applications for display at monitor **412**.

In conventional time-sharing networks that support large conventional databases, the host receives user requests for data records; locates them; and transmits them back to the users. Accordingly, the host is obliged to undertake the data processing necessary to isolate and supply the requested information. And, as noted earlier, where large numbers of users are to be served, the many user requests can bottleneck at the host, taxing resources and leading to response slowdown.

Further, users have experienced difficulty in searching data bases maintained on conventional time-sharing networks. For example, difficulties have resulted from the complex and varied way previously known database suppliers have organized and presented their information. Particularly, some database providers require searching be done only in selected fields of the data base, thus requiring the user to be fully familiar with the record structure. Others have organized their databases on hierarchial structures which require the user understand the way the records are grouped. Still further, yet other database suppliers rely upon keyword indices to facilitate searching of their records, thus requiring the user to be knowledgeable regarding the particular keywords used by the database provider.

Network **10**, however, is designed to avoid such difficulties. In the preferred embodiment, the network includes procedures for creating preliminary searches which represent subsets of the network applications users are believed likely to investigate. Particularly, in accordance with these procedures, for the active applications available on network **10**, a library of tables is prepared, and maintained within

each of which a plurality of so called "keywords" are provided that are correlated with page template objects and object-ids of the entry screen (typically the first screen) for the respective application. In the preferred embodiment, approximately 1,000 tables are used, each having approximately 10 to 20 keywords arranged in alphabetical order to abstract the applications on the network. Further, the object-id for each table is associated with a code in the form of a character string mnemonic which is arranged in a set of alphabetically sequenced mnemonics termed the sequence set so that on entry of a character string at an RS **400**, the object-id for the relevant keyword table can be obtained from the sequence set. Once the table object-id is identified, the keyword table corresponding to the desired subset of the objects and associated applications can then be obtained from network **10**. Subsequently the table can be presented to the user's RS **400**, where the RS **400** can provide the data processing required to present the potentially relevant keywords, objects and associated applications to the user for further review and determination as to whether more searching is required. As will be appreciated, this procedure reduces demand on server **205** and thereby permits it to be less complex and costly, and further, reduces the likelihood of host overtaxing that may cause network response slowdown.

As a further feature of this procedure, the library of keywords and their associated PTOs and objects may be generated by a plurality of operations which appear at the user's screen as different search techniques. This permits the user to select a search technique he is most comfortable with, thus expediting his inquiry.

More particularly, the user is allowed to invoke the procedure by calling up a variety of operations. The various operations have different names and seemingly present different search strategies. Specifically, the user may invoke the procedure by initiating a "Jump" command at RS **400**. Thereafter, in connection with the Jump operation, the user, when prompted, may enter a word of the user's choosing at monitor screen **414** relating to the matter he is interested in locating; i.e., a subject matter search of the network applications. Additionally, the users may invoke the procedure by alternatively calling up an operation termed "Index" with selection of the Index command. When selected, the Index command presents the user with an alphabetical listing of keywords from the tables noted above which the user can select from; i.e., an alphabetical search of the network applications. Further, the user may evoke the procedure by initiating an operation termed "Guide." By selecting the Guide command, the user is provided with a series of graphic displays that presents a physical description of the network applications; e.g., department floor plan for a store the user may be electronically shopping in. Still further, the user may invoke the procedures by initiating an operation termed "Directory." By selecting the Directory command, the user is presented with the applications available on the network as a series of hierarchial menus which present the content of the network information in commonly understood categories. Finally, the user may invoke the procedure by selecting the "Path" command, which accesses a list of keywords the user has previously selected; i.e., a personally tailored form of the Index command described above. As described hereafter, Path further includes a Viewpath operation which permits the user to visually access and manage the Path list of keywords. In preferred form, where the user has not selected a list of personalized keywords, a default set is provided which includes a predetermined list and associated applications deemed by network **10** as likely to be of interest to the user.

This ability to convert these apparently different search strategies in a single procedure for accessing pre-created library tables is accomplished by translating the procedural elements of the different search techniques into a single set of procedures that will produce a mnemonic; i.e., code word, which can first be searched at the sequence set, described above to identify the object-id for the appropriate library table and, thereafter, enable access of the appropriate table to permit selection of the desired keyword and associated PTO and object-ids. That is to say, the reception system native code simply relates the user-entered character string, alphabetical range, category, or list item of respectively, "Jump", "Index", "Directory", or "Path" to the table codes through the sequence set, so that the appropriate table can be provided to the reception system and application keyword selected. Thus, while the search techniques may appear different to the user, and in fact accommodate the user's preferences and sophistication level, they nonetheless invoke the same efficient procedure of relying upon pre-created searches which identify related application PTOs and object-ids so that the table and objects may be collected and presented at the user's RS **400** where they can be processed, thereby relieving server **205**.

In preferred form, however, in order to enhance presentation speed the Guide operation is specially configured. Rather than relating the keyword mnemonic to a sequence set to identify the table object-id and range of keywords corresponding to the entry PTO and associated object-ids, the Guide operation presents a series of overlapping windows that physically describe the "store" in which shopping is being conducted or the "building" from which information is being provided. The successive windows increase in degree of detail, with the final window presenting a listing of relevant keywords. Further, the PTO and object-ids for the application entry screen are directly related to the graphic presentation of the keywords. This eliminates the need to provide variable fields in the windows for each of the keywords and enables the entry screen to be correlated directly with the window graphic. As will be appreciated, this reduces the number of objects that would otherwise be required to be staged at RS **400** to support pretention of the keyword listing at monitor screen **414**, and thus speeds network response.

A more detailed understanding of the procedure may be had upon a reading of the following description and review of accompanying FIGS. **2**, **3***a* and particularly FIG. **11** which presents a flow diagram for the Jump sequence of the search procedure.

To select a particular partitioned application from among thousands of such applications residing either at the RS **400** or within delivery system **20**, network **10** avoids the need for a user to know or understand, prior to a search, the organization of such partitioned applications and the query techniques necessary to access them. This is accomplished using a collection of related commands, as described below.

The Jump command **296** as seen in FIG. **3***a*, can be selected, by the user from command bar **290**. When Jump command **296** is selected, a window partition **275** is opened. In window **275**, the user is presented and may select from a variety of displayed options that include among others, the Directory command, the Index command, and the Guide command, which when selected, have the effect noted above. Additionally, the user can select a command termed Viewpath which will presents the keywords that currently

make up the list of keywords associated with the user's Path command, and from which list the user can select a desired keyword. Still further, and with reference FIG. **11**, which shows the sequence where a user offers a term to identify a subject of interest, the user may enter a keyword at display field **270** within window partition **275** as a "best guess" of the mnemonic character string that is assigned to a partitioned application the user desires (e.g., the user may input such english words as "news," "pet food," "games," etcetera). Where the user enters a character string it is displayed in field **270**, and then searched by RS **400** native code (discussed below) against the sequence sets above noted to identify the object-id for the appropriate table of keywords (not shown) that RS **400** may request from host **205**. While as noted above, a table may include 10 to 20 keywords, in the preferred embodiment, for the sake of speed and convenience, a typical keyword table includes approximately 12 keywords.

If the string entered by the user matches a keyword existing on one of the keyword tables, and is thus associated with a specific PTO, RS **400** fetches and displays associated objects of the partitioned applications and builds the entry page in accordance with the page composition dictated by the target PTO.

If the string entered by the user does not match a specific keyword, RS **400** presents the user with the option of displaying the table of keywords approximating the specific keyword. The approximate keywords are presented as initialized, cursorable selector fields of the type provided in connection with a Index command. The user may then move the cursor to the nearest approximation of the mnemonic he originally selected, and trigger navigation to the PTO associated with that keyword, navigation being as described hereafter in connection with the RS **400** native code.

If, after selecting the Jump command, the user selects the Index command, RS **400** will retrieve the keyword table residing at RS **400**, and will again build a page with initialized, cursorable fields of keywords. The table fetched upon invoking the Index command will be comprised of alphabetic keywords that occur within the range of the keywords associated with the page template object (PTO) from which the user invoked the Index command. As discussed above, the user may select to navigate to any of this range of PTOs by selecting the relevant keyword from the display. Alternatively, the user can, thereafter, select another range of alphabetical keywords by entering an appropriate character string in a screen field provided or move forward or backward in the collection by selecting the corresponding option.

By selecting the Directory command, RS **400** can be caused to fetch a table of keywords, grouped by categories, to which the PTO of the current partitioned application (as specified by the object set field **630** of the current PEO) belongs. Particularly, by selecting the Directory command, RS **400**, is causes to displays a series of screens each of which contains alphabetically arranged general subject categories from which the user may select. Following selection of a category, a series of keywords associated with the specified category are displayed in further screens together with descriptive statements about the application associated with the keywords. Thereafter, the user can, in the manner previously discussed with regard to the Index command, select from and navigate to the PTOs of keywords which are related to the present page set by subject.

The Guide command provides a navigation method related to a hierarchical organization of applications provided on network **10**, and are described by a series of sequentially presented overlaying windows of a type known in the art, each of which presents an increasing degree of detail for a particular subject area, terminating in a final window that gives keywords associated with the relevant applications. The Guide command makes use of the keyword segment which describes the location of the PTO in a hierarchy (referred to, in the preferred embodiment, as the "BFD," or Building-Floor-Department) as well as an associated keyword character string. The BFD describes the set of menus that are to be displayed on the screen as the sequence of pop-up windows. The Guide command may be invoked by requesting it from the Jump window described above, or by selecting the Menu command on Command Bar **290**. As noted above, in the case of the Guide command, the PTO and object-ids for the application entry screen are directly associated with the graphic of the keyword presented in the final pop-up window. This enables direct access of the application entry screen without need to access the sequence set and keyword table, and thus, reduces response time by reducing the number of objects that must be processed at RS **400**.

Activation of the Path command accesses the user's list of pre-selected keywords without their display, and permits the user to step through the list viewing the respective applications by repeatedly invoking the Path command. As will be appreciated, the user can set a priority for selecting keywords and viewing their associated applications by virtue of where on the list the user places the keywords. More specifically, if the user has several application of particular interest; e.g., news, weather, etc., the user can place them at the top of the list, and quickly step through them with the Path command. Further, the user can view and randomly access the keywords of his list with the Viewpath operation noted above. On activation of Viewpath, the user's Path keywords are displayed and the user can cursor through them in a conventional manner to select a desired one. Further, the user can amend the list as desired by changing the keywords on the list and/or adjusting their relative position. This is readily accomplished by entering the amendments to the list presented at the screen **414** with a series of amendment options presented in a conventional fashion with the list. As noted, the list may be personally selected by the user in the manner described, or created as a default by network **10**.

Collectively, the Jump command, Index command, Directory command, Guide command, and Path command as described enable the user to quickly and easily ascertain the "location" of either the partitioned application presently displayed or the "location" of a desired partitioned application. "Location," as used in reference to the preferred embodiment means the specific relationships that a particular partitioned application bears to other such applications, and the method for selecting particular partitioned applications from such relationships. The techniques for querying a database of objects, embodied in network **10** is an advance over the prior art, insofar as no foreknowledge of either database structure or query technique or syntax is necessary, the structure and search techniques being made manifest to the user in the course of use of the commands.

RS Application Protocol

RS protocol defines the way the RS supports user application conversation (input and output) and the way RS **400** processes a partitioned application. Partitioned applications are constructed knowing that this protocol will be supported unless modified by the application. The protocol is illus-

trated FIG. **6**. The boxes in FIG. **6** identify processing states that the RS **400** passes through and the arrows indicate the transitions permitted between the various states and are annotated with the reason for the transition.

The various states are: (A) Initialize RS, (B) Process Objects, (C) Interpretively Execute Pre-processors, (D) Wait for Event, (E) Process Event, and (F) Interpretively Execute Function Extension and/or Post-processors.

The transitions between states are: (1a) Logon Page Template Object Identification (PTO-id), (1b) Object Identification, (2) Trigger Program Object identification (PO-id) & return, (3) Page Partition Template (PPT) or Window Stack Processing complete, (4) Event Occurrence, and (5) Trigger PO-id and Return.

Transition (1a) from Initialize RS (A) to Process Objects (B) occurs when an initialization routine passes the object-id of the logon PTO to object interpreter **435**, when the service is first invoked. Transition (1b) from Process Event (E) to Process Objects (B) occurs whenever a navigation event causes a new page template object identification (PTO-id) to be passed to object interpreter **435**; or when a open window event (verb or function key) occurs passing a window object-id to the object interpreter **435**; or a close window event (verb or function key) occurs causing the current top-most window to be closed.

While in the process object state, object interpreter **435** will request any objects that are identified by external references in call segments. Objects are processed by parsing and interpreting the object and its segments according to the specific object architecture. As object interpreter **435** processes objects, it builds a linked list structure called a page processing table (PPT), shown in FIG. **10**, to reflect the structure of the page, each page partition, Page Element Objects (PEOs) required, program objects (POs) required and each window object (WO) that could be called. Object interpreter **435** requests all objects required to build a page except objects that could be called as the result of some event, such as a HELP window object.

Transition (2) from Process Objects (B) to Interpretively Execute Pre-processors (C) occurs when the object interpreter **435** determines that a pre-processor is to be triggered. Object processor **436** then passes the object-id of the program object to the TBOL interpreter **438**. TBOL interpreter **438** uses the RS virtual machine to interpretively execute the program object. The PO can represent either a selector or an initializer. When execution is complete, a transition automatically occurs back to Process Objects (B).

Selectors are used to dynamically link and load other objects such as PEOs or other PDOs based upon parameters that they are passed when they are called. Such parameters are specified in call segments or selector segments. This feature enables RS **400** to conditionally deliver information to the user base upon predetermined parameters, such as his personal demographics or locale. For example, the parameters specified may be the transaction codes required to retrieve the user's age, sex, and personal interest codes from records contained in user profiles stored at the switch/file server layer **200**.

Initializers are used to set up the application processing environment for a partitioned application and determine what events RS **400** may respond to and what the action will be.

Transition (3) from Process Objects (B) to Wait for Event (D) occurs when object interpreter **435** is finished processing objects associated with the page currently being built or opening or closing a window on a page. In the Wait for Event state (D), an input manager, which in the preferred form shown includes keyboard manager **434** seen in FIG. **8**, accepts user inputs. All keystrokes are mapped from their physical codes to logical keystrokes by the Keyboard Manager **434**, representing keystrokes recognized by the RS virtual machine.

When the cursor is located in a field of a page element, keystrokes are mapped to the field and the partitioned external variable (PEV) specified in the page element object (PEO) field definition segment by the cooperative action of keyboard manager, **434** and display manager **461**. Certain inputs, such as RETURN or mouse clicks in particular fields, are mapped to logical events by keyboard manager **434**, which are called completion (or commit) events. Completion events signify the completion of some selection or specification process associated with the partitioned application and trigger a partition level and/or page level post-processor to process the "action" parameters associated with the user's selection and commit event.

Such parameters are associated with each possible choice or input, and are set up by the earlier interpretive execution of an initializer pre-processor in state (C). Parameters usually specify actions to perform a calculation such as the balance due on an order of several items with various prices using sales tax for the user's location, navigate to PTO-id, open window WO-id or close window. Actions parameters that involve the specification of a page or window object will result in transition (**1***b*) to the Process Objects (B) state after the post-processor is invoked as explained below.

Function keys are used to specify one or more functions which are called when the user strikes these keys. Function keys can include the occurrence of logical events, as explained above. Additionally, certain functions may be "filtered", that is, extended or altered by SET_FUNCTION or TRIGGER_FUNCTION verbs recognized by the RS virtual machine. Function keys cause the PO specified as a parameter of the verb to be interpretively executed whenever that function is called. Applications use this technique to modify or extend the functions provided by the RS.

Transition (5) from Process Event (E) to Interpretively Execute Pre-processors (F) occurs when Process Event State determines that a post-processor or function extension PDO is to be triggered. The id of the program object is then passed to the TBOL interpreter **438**. The TBOL interpreter **438** uses the RS virtual machine to interpretively execute the PO. When execution is complete a transition automatically occurs back to Process Event (E).

Reception System Software

The reception system **400** software is the interface between the user of personal computer **405** and interactive network **10**. The object of reception system software is to minimize mainframe processing, minimize transmission across the network, and support application extendibility and portability.

RS **400** software is composed of several layers, as shown in FIG. **7**. It includes external software **451**, which is composed of elements well known to the art such as device drivers, the native operating systems; i.e., MS-DOS, machine-specific assembler functions (in the preferred embodiment; e.g., CRC error checking), and "C" runtime library functions; native software **420**; and partitioned applications **410**.

Again with reference to FIG. **7**, native software **420** is compiled from the "C" language into a target machine-specific executable, and is composed of two components: the service software **430** and the operating environment **450**.

Operating environment **450** is comprised of the Logical Operating System **432**, or LOS; and a multitasker **433**. Service software **430** provides functions specific to providing interaction between the user and interactive network **10**, while the operating environment **450** provides pseudo multitasking and access to local physical resources in support of service software **430**. Both layers of native software **420** contain kernel, or device independent functions **430** and **432**, and machine-specific or device dependent functions **433**. All device dependencies are in code resident at RS **400**, and are limited to implementing only those functions that are not common across machine types, to enable interactive network **10** to provide a single data stream to all makes of personal computer which are of the IBM or IBM compatible type. Source code for the native software **420** is included in parent application Ser. No. 388,156 now issued as U.S. Pat. No. 5,347,632, the contents of which patent are incorporated herein by reference. Those interested in a more detailed description of the reception system software may refer to the source code provided in the referenced patent.

Service software **430** is comprised of modules, which are device-independent software components that together obtain, interpret and store partitioned applications existing as a collection of objects. The functions performed by, and the relationship between, the service software **430** module is shown in FIG. **8** and discussed further below.

Through facilities provided by LOS **432** and multitasker **433**, here called collectively operating environment **450**, device-independent multitasking and access to local machine resources, such as multitasking, timers, buffer management, dynamic memory management, file storage and access, keyboard and mouse input, and printer output are provided. The operating environment **450** manages communication and synchronization of service software **430**, by supporting a request/response protocol and managing the interface between the native software **420** and external software **437**.

Applications software layer **410** consists of programs and data written in an interpretive language, "TRINTEX Basic Object Language" or "TBOL," described above. TBOL was written specifically for use in RS **400** and interactive network **10** to facilitate videotext-specific commands and achieve machine-independent compiling. TBOL is constructed as objects, which in interaction with one another comprise partitioned applications.

RS native software **420** provides a virtual machine interface for partitioned applications, such that all objects comprising partitioned applications "see" the same machine. RS native software provides support for the following functions: (1) keyboard and mouse input; (2) text and graphics display; (3) application interpretation; (4) application database management; (5) local application storage; (6) network and link level communications; (7) user activity data collection; and (8) advertisement management.

With reference to FIG. **8**, service software **430** is comprised of the following modules: start-up (not shown); keyboard manger **434**; object interpreter **435**; TBOL interpreter **438**; object storage facility **439**; display manager **461**; data collection manager **441**; ad manager **442**; object/communications manager interface **443**; link communications manager **444**; and fatal error manager **469**. Each of these modules has responsibility for managing a different aspect of RS **400**.

Startup reads RS **400** customization options into RAM, including modem, device driver and telephone number options, from the file CONFIG.SM. Startup invokes all RS **400** component startup functions, including navigation to the first page, a logon screen display containing fields initialized to accept the user's id and password. Since Startup is invoked only at initialization, for simplicity, it has not been shown in FIG. **8**.

The principal function of keyboard manger **434** is to translate personal computer dependent physical input into a consistent set of logical keys and to invoke processors associated with these keys. Depending on the LOS key, and the associated function attached to it, navigation, opening of windows, and initiation of filter or post-processor TBOL programs may occur as the result input events handled by the keyboard manger **434**. In addition, keyboard manger **434** determines inter and intra field cursor movement, and coordinates the display of field text and cursor entered by the user with display manager **461**, and sends information regarding such inputs to data collection manager **441**.

Object interpreter **435** is responsible for building and recursively processing a table called the "Page Processing Table," or PPT. Object interpreter **435** also manages the opening and closing of windows at the current page. Object interpreter **435** is implemented as two sub-components: the object processor **436** and object scanner **437**.

Object processor **436** provides an interface to keyboard manger **434** for navigation to new pages, and for opening and closing windows in the current page. Object processor **436** makes a request to object storage facility **439** for a page template object (PTO) or window object (WO), as requested by keyboard manger **434**, and for objects and their segments which comprise the PTO or WO returned by object storage facility **439** to object processor **436**. Based on the particular segments comprising the object(s) making up the new PTO or WO, object processor **436** builds or adds to the page processing table (PPT), which is an internal, linked-list, global data structure reflecting the structure of the page or page format object (PFO), each page partition or page element object (PEO), and program objects (POs) required and each window object (WO) that could be called. Objects are processed by parsing and interpreting each object and its-segment(s) according to their particular structure as formalized in the data object architecture (DOA). While in the process object state, (state "B" of FIG. **6**), object processor **436** will request any objects specified by the PTO that are identified by external references in call segments (e.g. field level program call **518**, page element selector call **524**, page format call **526** program call **532**, page element call **522** segments) of such objects, and will, through a request to TBOL interpreter **438**, fire initializers and selectors contained in program data segments of all PTO constituent program objects, at the page, element, and field levels. Object processor **436** requests all objects required to build a page, except objects that could only be called as the result of some event external to the current partitioned application, such as a HELP window object. When in the course of building or adding to the PPT and opening/closing WOs, object processor encounters a call to an "ADSLOT" object id, the next advertisement object id at ad manager **442** is fetched, and the identified advertisement object is retrieved either locally, if available, or otherwise from the network, so that the presentation data for the advertisement can be sent to display manager **461** along with the rest of the presentation data for the other objects to enable display to the user. Object processor **436** also passes to data collection manager **441** all object ids that were requested and object ids that were viewed. Upon completion of page or window processing, object processor **436** enters the wait for event state, and control is returned to keyboard manger **434**.

The second component of object interpreter **435**, object scanner **437**, provides a file-like interface, shared with object storage facility **439**, to objects currently in use at RS **400**, to enable object processor **436** to maintain and update the PPT. Through facilities provided by object scanner **437**, object processor recursively constructs a page or window in the requested or current partitioned application, respectively.

Object storage facility **439** provides an interface through which object interpreter **435** and TBOL interpreter **438** either synchronously request (using the TBOL verb operator "GET") objects without which processing in either module cannot continue, or asynchronously request (using the TBOL verb operator "FETCH") objects in anticipation of later use. Object storage facility **439** returns the requested objects to the requesting module once retrieved from either local store **440** or interactive network **10**. Through control structures shared with the object scanner **437**, object storage facility determines whether the requested object resides locally, and if not, makes an attempt to obtain it from interactive network **10** through interaction with link communications manager **444** via object/communications manager interface **443**.

When objects are requested from object storage facility **439**, only the latest version of the object will be provided to guarantee currency of information to the user. Object storage facility **439** assures currency by requesting version verification from network **10** for those objects which are available locally and by requesting objects which are not locally available from delivery system **20** where currency is maintained.

Version verification increases response time. Therefore, not all objects locally available are version checked each time they are requested. Typically, objects are checked only the first time they are requested during a user session. However, there are occasions, as for example in the case of objects relating to news applications, where currency is always checked to assure integrity of the information.

The frequency with which the currency of objects is checked depends on factors such as the frequency of updating of the objects. For example, objects that are designated as ultrastable in a storage control parameter in the header of the object are never version checked unless a special version control object sent to the RS as part of logon indicates that all such objects must be version checked. Object storage facility **439** marks all object entries with such a stability category in all directories indicating that they must be version checked the next time they are requested.

Object storage facility **439** manages objects locally in local store **440**, comprised of a cache (segmented between available RAM and a fixed size disk file), and stage (fixed size disk file). Ram and disk cached objects are retained only during user sessions, while objects stored in the stage file are retained between sessions. The storage control field, located in the header portion of an object, described more fully hereafter as the object "storage candidacy", indicates whether the object is stageable, cacheable or trashable.

Stageable objects must not be subject to frequent change or update. They are retained between user sessions on the system, provided storage space is available and the object has not discarded by a least-recently-used (LRU) algorithm of a conventional type; e.g., see *Operating System Theory*, by Coffman, Jr. and Denning, Prentice Hall Publishers, New York, 1973, which in accordance with the design of network **10**, operates in combination with the storage candidacy value to determine the object storage priority, thus rendering the stage self-configuring as described more fully hereafter. Over time, the self-configuring stage will have the effect of retaining within local disk storage those objects which the user has accessed most often. The objects retained locally are thus optimized to each individual user's usage of the applications in the system. Response time to such objects is optimized since they need not be retrieved from the interactive computer system.

Cacheable objects can be retained during the current user session, but cannot be retained between sessions. These objects usually have a moderate update frequency. Object storage facility **439** retains objects in the cache according to the LRU storage retention algorithm. Object storage facility **439** uses the LRU algorithm to ensure that objects that are least frequently used forfeit their storage to objects that are more frequently used.

Trashable objects can be retained only while the user is in the context of the partitioned application in which the object was requested. Trashable objects usually have a very high update frequency and must not be retained to ensure that the user has access to the most current data.

More particularly and, as noted above, in order to render a public informational and transactional network of the type considered here attractive, the network must be both economical to use and fast. That is to say, the network must supply information and transactional support to the user at minimal costs and with a minimal response time. These objectives are sought to be achieved by locating as many information and transactional support objects which the user is likely to request, as close to the user as possible; i.e., primarily at the user's RS **400** and secondarily at delivery system **20**. In this way, the user will be able to access objects required to support a desired application with minimal intervention of delivery system **20**, thus reducing the cost of the session and speeding the response time.

However, the number of objects that can be maintained at RS **400** is restricted by at least two factors: the RS **400** storage capacity; i.e., RAM and disk sizes, and the need to maintain the stored objects current.

In order to optimize the effectiveness of the limited storage space at RS **400**, the collection of objects is restricted to those likely to be requested by the user; i.e., tailored to the user's tastes—and to those least likely to be time sensitive; i.e., objects which are stable. To accomplish this, objects are coded for storage candidacy to identify when they will be permitted at RS **400**, and subject to the LRU algorithm to maintain presence at RS **400**. Additionally, to assure currency of the information and transaction support provided at RS **400**, objects are further coded for version identification and checking in accordance with a system of priorities that are reflected in the storage candidacy coding.

Specifically, to effect object storage management, objects are provided with a coded version id made up of the storage control byte and version control bytes identified above as elements of the object header, specifically, bytes **16** and **18** shown in FIG. **4***b*. In preferred form, the version id is comprised of bytes **16** and **18** to define two fields, a first 13 bit field to identify the object version and a second three bite field to identify the object storage candidacy.

In this arrangement, the storage candidacy value of the object is addressed to not only the question of storage preference but also object currency. Specifically, the storage candidacy value establishes the basis upon which the object will be maintained at RS **400** and also identifies the susceptibility of the object to becoming stale by dictating when the object will be version checked to determine currency.

The version value of the object on the other hand, provides a parameter that can be checked against predetermined values available from delivery system **20** to deter-

mine whether an object stored at RS **400** is sufficiently current to permit its continued use, or whether the object has become stale and needs to be replaced with a current object from delivery system **20**.

Still further, object storage management procedure further includes use of the LRU algorithm, for combination with the storage and version coding to enable discarding of objects which are not sufficiently used to warrant retention, thus personalizing the store of objects at RS **400** to the user's tastes. Particularly, object storage facility **439**, in accordance with the LRU algorithm maintains a usage list for objects. As objects are called to support the user's applications requests, the objects are moved to the top of a usage list. As other objects are called, they push previously called objects down in the list. If an object is pushed to the bottom of the list before being recalled, it will be forfeited from the list if necessary to make room for the next called object. As will be appreciated, should a previously called object be again called before it is displaced from the list, it will be promoted to the top of the list, and once more be subject to depression in the list and possible forfeiture as other objects are called.

As pointed out above, in the course of building the screens presented to the user, objects will reside at various locations in RS **400**. For example, objects may reside in the RS **400** RAM where the object is supporting a particular application screen then running or in a cache maintained at either RAM or disk **424** where the object is being held for an executing application or staged on the fixed size file on disk **424** noted above where the object is being held for use in application likely to be called by the user in the future.

In operation, the LRU algorithm is applied to all these regions and serves to move an object from RAM cache to disk cache to disk file, and potentially off RS **400** depending on object usage.

With regard to the storage candidacy value, in this arrangement, the objects stored at RS **400** include a limited set of permanent objects; e.g., those supporting logon and logoff, and other non-permanent objects which are subject to the LRU algorithm to determine whether the objects should be forfeited from RS **400** as other objects are added. Thus, in time, and based on the operation of the LRU algorithm and the storage candidacy value, the collection of objects at RS **400** will be tailored to the usage characteristics of the subscriber; i.e., self-configuring.

More particularly, the 3-bit field of the version id that contains the storage candidacy parameter can have 8 different values. A first candidacy value is applied where the object is very sensitive to time; e.g., news items, volatile pricing information such as might apply to stock quotes, etc. In accordance with this first value, the object will not be permitted to be stored on RS **400**, and RS **400** will have to request such objects from delivery system **20** each time it is accessed, thus, assuring currency. A second value is applied where the object is sensitive to time but less so than the first case; e.g., the price of apples in a grocery shopping application. Here, while the price might change from day to day, it is unlikely to change during a session. Accordingly the object will be permitted to persist in RAM or at the disk cache during a session, but will not be permitted to be maintained at RS **400** between sessions.

Continuing down the hierarchy of time sensitivity, where the object concerns information sufficiently stable to be maintained between sessions, a third storage candidacy value is set to permit the object to be stored at RS **400** between sessions, on condition that the object will be version check the first time it is accessed in a subsequent session. As will be appreciated, during a session, and under the effect of the LRU algorithm, lack of use at RS **400** of the object may result in it being forfeited entirely to accommodate new objects called for execution at RS **400**.

Still further, a fourth value of storage candidacy is applied where the object is considered sufficiently stable as not to require version checking between sessions; e.g., objects concerning page layouts not anticipated to change. In this case, the storage candidacy value may be encoded to permit the object to be retained from session to session without version checking. Here again, however, the LRU algorithm may cause the object to forfeit its storage for lack of use.

Where the object is of a type required to be stored at RS **400**, as for example, objects needed to support standard screens, it is coded for storage between sessions and not subject to the LRU algorithm forfeiture. However, where such objects are likely to change in the future they may be required to be version checked the first time they are accessed in a session and thus be given a fifth storage candidacy value. If, on the other hand, the required stored object is considered likely to be stable and not require even version checking; e.g., logon screens, it will be coded with a sixth storage candidacy value for storage without version checking so as to create a substantially permanent object.

Continuing, where a RS **400** includes a large amount of combined RAM and disk capacity, it would permit more objects to be stored. However, if objects were simply coded in anticipation of the larger capacity, the objects would potentially experience difficulty, as for example, undesired forfeiture due to capacity limitations if such objects were supplied to RS **400** units having smaller RAM and disk sizes. Accordingly, to take advantage of the increased capacity of certain RS **400** units without creating difficulty in lower capacity units, objects suitable for storage in large capacity units can be so coded for retention between sessions with a seventh and eighth storage candidacy value depending upon whether the stored large capacity object requires version checking or not. Here, however, the coding will be interpreted by smaller capacity units to permit only cacheable storage to avoid undesirable forfeiture that might result from over filling the smaller capacity units.

Where an object is coded for no version checking need may nonetheless arise for a version check at some point. To permit version checking of such objects, a control object is provided at RS **400** that may be version checked on receipt of a special communication from delivery system **20**. If the control object fails version check, then a one shot version checking attribute is associated with all existing objects in RS **400** that have no version checking attributes. Thereafter, the respective objects are version checked, the one shot check attribute is removed and the object is caused to either revert to its previous state if considered current or be replaced if stale.

Still further, objects required to be stored at RS **400** which are not version checked either because of lack of requirement or because of no version check without a control object, as described above, can accumulate in RS **400** as dead objects. To eliminate such accumulation, all object having required storage are version checked over time. Particularly, the least recently used required object is version checked during a session thus promoting the object to the top of the usage list if it is still to be retained at RS **400**. Accordingly, one such object will be checked per session and over time, all required objects will be version checked thereby eliminating the accumulation of dead objects.

However, in order to work efficiently, the version check attribute of the object should be ignored, so that even required object can be version checked. Yet, in certain

circumstances, e.g., during deployment of new versions of the reception system software containing new objects not yet supported on delivery system **20** which may be transferred to the fixed storage file of RS **400** when the new version is loaded, unconditional version checking may prematurely deletes the object from the RS **400** as not found on delivery system **20**. To avoid this problem, a sweeper control segment in the control object noted above can be used to act as a switch to turn the sweep of dead objects on and off.

With respect to version checking for currency, where an object stored at RS **400** is initially fetched or accessed during a session, a request to delivery system **20** is made for the object by specifying the version id of the object stored at RS **400**.

In response, delivery system **20** will advise the reception system **400** either that the version id of the stored object matches the currency value; i.e., the stored object is acceptable, or deliver a current object that will replace the stored object shown to be stale. Alternatively, the response may be that the object was not found. If the version of the stored object is current, the stored object will be used until verified again in accordance with its storage candidacy. If the stored object is stale, the new object delivered will replace the old one and support the desired screen. If the response is object not found, the stored object will be deleted.

Therefore, based on the above description, network **10** is seen to include steps for execution at storage facility **439** which enables object reception, update and deletion by means of a combination of operation of the LRU algorithm and interpretation of the storage candidacy and version control values. In turn, these procedures cooperate to assure a competent supply of objects at RS **400** so as to reduce the need for intervention of delivery system **20**, thus reducing cost of information supply and transactional support so as to speed the response to user requests.

TBOL interpreter **438** shown in FIG. **8** provides the means for executing program objects, which have been written using an interpretive language, TBOL described above. TBOL interpreter **438** interprets operators and operand contained in program object **508**, manages TBOL variables and data, maintains buffer and stack facilities, and provides a runtime library of TBOL verbs.

TBOL verbs provide support for data processing, program flow control, file management, object management, communications, text display, command bar control, open/close window, page navigation and sound. TBOL interpreter also interacts with other native modules through commands contained in TBOL verbs. For example: the verb "navigate" will cause TBOL interpreter **438** to request object interpreter **435** to build a PPT based on the PTO id contained in the operand of the NAVIGATE verb; "fetch" or "GET" will cause TBOL interpreter **438** to request an object from object storage facility **439**; "SET_FUNCTION" will assign a filter to events occurring at the keyboard manger **434**; and "FORMAT," "SEND," and "RECEIVE" will cause TBOL interpreter **438** to send application level requests to object/communications manager interface **433**.

Data areas managed by TBOL interpreter **438** and available to TBOL programs are Global External Variables (GEVs), Partition External Variables (PEVs), and Runtime Data Arrays (RDAs).

GEVs contain global and system data, and are accessible to all program objects as they are executed. GEVs provide a means by which program objects may communicate with other program objects or with the RS native code, if declared in the program object. GEVs are character string variables that take the size of the variables they contain. GEVs may preferably contain a maximum of 32,000 variables and are typically used to store such information as program return code, system date and time, or user sex or age. TBOL interpreter **438** stores such information in GEVs when requested by the program which initiated a transaction to obtain these records from the RS or user's profile stored in the interactive system.

Partition external variables (PEVs) have a scope restricted to the page partition on which they are defined. PEVs are used to hold screen field data such that when PEOs and window objects are defined, the fields in the page partitions with which these objects are to be associated are each assigned to a PEV. When applications are executed, TBOL interpreter **438** transfers data between screen fields and their associated PEV. When the contents of a PEV are modified by user action or by program direction, TBOL interpreter **428** makes a request to display manager **461** to update the screen field to reflect the change. PEVs are also used to hold partition specific application data, such as tables of information needed by a program to process an expected screen input.

Because the scope of PEVs is restricted to program objects associated with the page partition in which they are defined, data that is to be shared between page partitions or is to be available to a page-level processor must be placed in GEVs or RDAs.

RDAs are internal stack and save buffers used as general program work areas. RDAs are dynamically defined at program object "runtime" and are used for communication and transfer of data between programs when the data to be passed is not amenable to the other techniques available. Both GEVs and RDAs include, in the preferred embodiment, 8 integer registers and 8 decimal registers. Preferably, there are also 9 parameter registers limited in scope to the current procedure of a program object.

All variables may be specified as operand of verbs used by the virtual machine. The integer and decimal registers may be specified as operand for traditional data processing. The parameter registers are used for passing parameters to "called" procedures. The contents of these registers are saved on an internal program stack when a procedure is called, and are restored when control returns to the "calling" procedure from the "called" procedure.

TBOL interpreter **438**, keyboard manger **434**, object interpreter **435**, and object storage facility **439**, together with device control provided by operating environment **450**, have principal responsibility for the management and execution of partitioned applications at the RS **400**. The remaining native code modules function in support and ancillary roles to provide RS **400** with the ability display partitioned applications to the user (display manager **461**), display advertisements (ad manager **442**), to collect usage data for distribution to interactive network **10** for purposes of targeting such advertisements (data collection manager **441**), and prepare for sending, and send, objects and messages to interactive network **10** (object/communications manager interface **443** and link communications manager **444**) Finally, the fatal error manager exists for one purpose: to inform the user of RS **400** and transmit to interactive network **10** the inability of RS **400** to recover from a system error.

Display manager **461** interfaces with a decoder using the North American Presentation Level Protocol Syntax (NAPLPS), a standard for encoding graphics data, or text code, such as ASCII, which are displayed on monitor **412** of the user's personal computer **405** as pictorial codes. Codes for other presentation media, such as audio, can be specified by

using the appropriate type code in the presentation data segments. Display manager **461** supports the following functions: send NAPLPS strings to the decoder; echo text from a PEV; move the cursor within and between fields; destructive or non-destructive input field character deletion; "ghost" and "unghost" fields (a ghosted field is considered unavailable, unghosted available); turn off or on the current field cursor; open, close, save and restore bit maps for a graphics window; update all current screen fields by displaying the contents of their PEVs, reset the NAPLPS decoder to a known state; and erase an area of the screen by generating and sending NAPLPS to draw a rectangle over that area. Display manager **461** also provides a function to generate a beep through an interface with a machine-dependent sound driver.

In accordance with the method of the present invention, Ad manager **442** is invoked by object interpreter **435** to return the object id of the next available advertisement to be displayed. Ad manager **442** maintains a queue of advertising object id's targeted to the specific user currently accessing interactive network **10**. Advertising objects are pre-fetched from interactive system **10** from a personalized queue of advertising ids that is constructed using data previously collected from user generated events and/or reports of objects used in the building of pages or windows, compiled by data collection manager **466** and transmitted to interactive system **10**.

Advertising objects **510** are PEOs that, through user invocation of a "LOOK" command, cause navigation to partitioned applications that may themselves support, for example, ordering and purchasing of merchandise.

An advertising object id list, or "ad queue," is requested in a transaction message to delivery system **20** by ad manager **442** immediately after the initial logon response. The logon application at RS **400** places the advertising list in a specific RS global storage area called a SYS_GEV (system global external variable), which is accessible to all applications as well as to the native RS code). The Logon application also obtains the first two ad object id's from the queue and provides them to object storage facility **439** so the advertising objects can be requested. However, at logon, since no advertising objects are available at RS local storage facilities **440**, ad objects, in accordance with the described storage candidacy, not being retained at the reception system between sessions, they must be requested from interactive network **10**.

In a preferred embodiment, the following parametric values are established for ad manager **442**: advertising object is queue capacity, replenishment threshold for advertising object id's and replenishment threshold for number of outstanding pre-fetched advertising objects. These parameters are set up in GEVs of the RS virtual machine by the logon application program object from the logon response from high function system **110**. The parameters are then also accessible to the ad manager **442**. Preferred values are an advertising queue capacity of 15, replenishment value of 10 empty queue positions and a pre-fetched advertising object threshold of 3.

Ad manager **442** pre-fetches advertising objects by passing advertising object id's from the advertising queue to object storage facility **439** which then retrieves the object from the interactive system if the object is not available locally. Advertising objects are pre-fetched, so they are available in RS local store **440** when requested by object interpreter **435** as it builds a page. The ad manager **442** pre-fetches additional advertising objects whenever the number of pre-fetched advertising objects not called by object interpreter **435**; i.e. the number of remaining advertising objects, falls below the pre-fetch advertising threshold.

Whenever the advertising object id queue has more empty positions than replenishment threshold value, a call is made to the advertising object id queue application in high function system **110** shown in FIG. **2**, via object/communications manager interface **443** for a number of advertising object id's equal to the threshold value. The response message from system **110** includes a list of advertising object id's, which ad manager **442** enqueues.

Object interpreter **435** requests the object id of the next advertising object from ad manager **442** when object interpreter **435** is building a page and encounters an object call for a partition and the specified object-id equals the code word, "ADSLOT." If this is the first request for an advertising object id that ad manager **442** has received during this user's session, ad manager **442** moves the advertising object id list from the GEV into its own storage area, which it uses as an advertising queue and sets up its queue management pointers, knowing that the first two advertising objects have been pre-fetched.

Ad manager **442** then queries object storage facility **439**, irrespective of whether it was the first request of the session. The query asks if the specified advertising object id pre-fetch has been completed, i.e., is the object available locally at the RS. If the object is available locally, the object-id is passed to object interpreter **435**, which requests it from object storage facility **439**. If the advertising object is not available in local store **440**, ad manager **442** attempts to recover by asking about the next ad that was pre-fetched. This is accomplished by swapping the top and second entry in the advertising queue and making a query to object storage facility **439** about the new top advertising object id. If that object is not yet available, the top position is swapped with the third position and a query is made about the new top position.

Besides its ability to provide advertising that have been targeted to each individual user, two very important response time problems have been solved by ad manager **442** of the present invention. The first is to eliminate from the new page response time the time it takes to retrieve an advertising object from the host system. This is accomplished by using the aforementioned pre-fetching mechanism.

The second problem is caused by pre-fetching, which results in asynchronous concurrent activities involving the retrieval of objects from interactive system **10**. If an advertising object is pre-fetched at the same time as other objects required for a page are requested, the transmission of the advertising object packets could delay the transmission of the other objects required to complete the current page by the amount of time required to transmit the advertising object(s). This problem is solved by the structuring the requests from object interpreter **435** to the ad manager **442** in the following way:

1. Return next object id of pre-fetched advertising object & pre-fetch another;
2. Return next advertising object id only; and
3. Pre-fetch next advertising object only.

By separating the function request (1) into its two components, (2) and (3), object interpreter **435** is now able to determine when to request advertising object id's and from its knowledge of the page build process, is able to best determine when another advertising object can be pre-fetched, thus causing the least impact on the page response time. For example, by examining the PPT, object interpreter **435** may determine whether any object requests are out-

standing. If there are outstanding requests, advertising request type 2 would be used. When all requested objects are retrieved, object interpreter **435** then issues an advertising request type 3. Alternatively, if there are no outstanding requests, object interpreter **435** issues an advertising request type 1. This typically corresponds to the user's "think time" while examining the information presented and when RS **400** is in the Wait for Event state (D).

Data collection manager **441** is invoked by object interpreter **435** and keyboard manger **434** to keep records about what objects a user has obtained (and, if a presentation data segment **530** is present, seen) and what actions users have taken (e.g. "NEXT," "BACK," "LOOK," etc.)

The data collection events that are to be reported during the user's session are sensitized during the logon process. The logon response message carries a data collection indicator with bit flags set to "on" for the events to be reported. These bit flags are enabled (on) or disabled (off) for each user based on information contained in the user's profile stored and sent from high function host **110**. A user's data collection indicator is valid for the duration of his session. The type of events to be reported can be changed at will in the host data collection application. However, such changes will affect only users who logon after the change.

Data collection manager **441** gathers information concerning a user's individual system usage characteristics. The types of informational services accessed, transactions processed, time information between various events, and the like are collected by data collection manager **441**, which compiles the information into message packets (not shown). The message packets are sent to network **10** via object/communication manager interface **443** and link communications manager **444**. Message packets are then stored by high function host **110** and sent to an offline processing facility for processing. The characteristics of users are ultimately used as a means to select or target various display objects, such as advertising objects, to be sent to particular users based on consumer marketing strategies, or the like, and for system optimization.

Object/communications manager interface **443** is responsible for sending and receiving DIA (Data Interchange Architecture described above) formatted messages to or from interactive network **10**. Object/communications manager **443** also handles the receipt of objects, builds a DIA header for messages being sent and removes the header from received DIA messages or objects, correlates requests and responses, and guarantees proper block sequencing. Object/communications manager interface **443** interacts with other native code modules as follows: object/communications manager **443** (1) receives all RS **400** object requests from object storage facility **439**, and forwards objects received from network **10** via link communications manager **444** directly to the requesting modules; (2) receives ad list requests from ad manager **442**, which thereafter periodically calls object/communications manager **443** to receive ad list responses; (3) receives data collection messages and send requests from data collection manager **441**; (4) receives application-level requests from TBOL interpreter **438**, which also periodically calls object/communications manager interface **443** to receive responses (if required); and (5) receives and sends DIA formatted objects and messages from and to link communications manager **444**.

Object/communications manager interface **443** sends and receives DIA formatted messages on behalf of TBOL interpreter **438** and sends object requests and receives objects on behalf of object storage facility **439**. Communication packets received containing parts of requested objects are passed to object storage facility **439** which assembles the packets into the object before storing it. If the object was requested by object interpreter **435**, all packets received by object storage facility **439** are also passed to object interpreter **435** avoiding the delay required to receive an entire object before processing the object. Objects which are pre-fetched are stored by object storage facility **439**.

Messages sent to interactive network **10** are directed via DIA to applications in network **10**. Messages may include transaction requests for records or additional processing of records or may include records from a partitioned application program object or data collection manager **441**. Messages to be received from network **10** usually comprise records requested in a previous message sent to network **10**. Requests received from object storage facility **439** include requests for objects from storage in interactive system **10**. Responses to object requests contain either the requested object or an error code indicating an error condition.

Object/communications manager **443** is normally the exclusive native code module to interface with link communications manager **444** (except in the rare instance of a fatal error). Link communications manager **444** controls the connecting and disconnecting of the telephone line, telephone dialing, and communications link data protocol. Link communications manager **444** accesses network **10** by means of a communications medium (not shown) link communications manager **444**, which is responsible for a dial-up link on the public switched telephone network (PSTN). Alternatively, other communications means, such as cable television or broadcast media, may be used. Link communications manager **444** interfaces with TBOL interpreter for connect and disconnect, and with interactive network **10** for send and receive.

Link communications manager **444** is subdivided into modem control and protocol handler units. Modem control (a software function well known to the art) hands the modem specific handshaking that occurs during connect and disconnect. Protocol handler is responsible for transmission and receipt of data packets using the TCS (TRINTEX Communications Subsystem) protocol (which is a variety of OSI link level protocol, also well known to the art).

Fatal error manager **469** is invoked by all reception system components upon the occurrence of any condition which precludes recovery. Fatal error manager **469** displays a screen to the user with a textual message and an error code through display manager **461**. Fatal error manager **469** sends an error report message through the link communications manager **444** to a subsystem of interactive network **10**.

The source code for the reception system software as noted above is described in parent application Ser. No. 388,156 filed Jul. 28, 1989, now issued as U.S. Pat. No. 5,347,632, the contents of which are incorporated herein by reference.

Sample Application

Page **255** illustrated in FIG. **3***b* corresponds to a partitioned application that permit's a user to purchase apples. It shows how the monitor screen **414** of the reception system **400** might appear to the user. Displayed page **255** includes a number of page partitions and corresponding page elements.

The page template object (PTO) **500** representing page **255** is illustrated in FIG. **9**. PTO **500** defines the composition of the page, including header **250**, body **260**, display fields **270**, **271**, **272**, advertising **280**, and command bar **290**. Page element objects (PEOs) **504** are associated with page parti-

tions numbered; e.g., **250**, **260**, **280**. They respectively, present information in the header **250**, identifying the page topic as ABC APPLES; in the body **260**, identifying the cost of apples; and prompt the user to input into fields within body **260** the desired number of apples to be ordered. In advertising **280**, presentation data and a field representing a post-processor that will cause the user to navigate to a targetable advertising, is presented.

In FIG. **9**, the structure of PTO **500** can be traced. PTO **500** contains a page format call segment **526**, which calls page format object (PFO) **502**. PFO **502** describes the location and size of partitions on the page and numbers assigned to each partition. The partition number is used in page element call segments **522** so that an association is established between a called page element object (PEO) **504** and the page partition where it is to be displayed. Programs attached to this PEO can be executed only when the cursor is in the page partition designated within the PEO.

PTO **500** contains two page element call segments **522**, which reference the PEOs **504** for partitions **250** and **260**. Each PEO **504** defines the contents of the partition. The header in partition **250** has only a presentation data segment **530** in its PEO **504**. No input, action, or display fields are associated with that partition.

The PEO **504** for partition **260** contains a presentation data segment **530** and field definition segments **516** for the three fields that are defined in that partition. Two of the fields will be used for display only. One field will be used for input of user supplied data.

In the example application, the PEO **504** for body partition **260** specifies that two program objects **508** are part of the body partition. The first program, shown in Display field **270**, **271**, **272**, is called an initializer and is invoked unconditionally by TBOL interpreter **438** concurrently with the display of presentation data for the partition. In this application, the function of the initializer is represented by the following pseudo-code:

1. Move default values to input and display fields;
2. "SEND" a transaction to the apple application that is resident on interactive system **10**;
3. "RECEIVE" the result from interactive system **10**; i.e. the current price of an apple;
4. Move the price of an apple to PEV **271** so that it will be displayed;
5. Position the cursor on the input field; and
6. Terminate execution of this logic.

The second program object **508** is a field post-processor. It will be invoked conditionally, depending upon the user keystroke input. In this example, it will be invoked if the user changes the input field contents by entering a number. The pseudo code for this post-processor is as follows:

1. Use the value in PEV **270** (the value associated with the data entered by the user into the second input data field **270**) to be the number of apples ordered.
2. Multiply the number of apples ordered times the cost per apple previously obtained by the initializer;
3. Construct a string that contains the message "THE COST OF THE APPLES YOU ORDERED IS $45.34;";
4. Move the string into PEV **272** so that the result will be displayed for the user; and
5. Terminate execution of this logic.

The process by which the "APPLES" application is displayed, initialized, and run is as follows.

The "APPLES" application is initiated when the user navigates from the previous partitioned application, with the navigation target being the object id of the "APPLES" PTO **500** (that is, object id ABC**1**). This event causes keyboard manager **434** to pass the PTO object id, ABC**1** (which may, for example, have been called by the keyword navigation segment **520** within a PEO **504** of the previous partitioned application), to object interpreter **435**. With reference to the RS application protocol depicted in FIG. **6**, when the partitioned application is initiated, RS **400** enters the Process Object state (B) using transition (1). Object interpreter **435** then sends a synchronous request for the PTO **500** specified in the navigation event to object storage facility **439**. Object storage facility **439** attempts to acquire the requested object from local store **440** or from delivery system **20** by means of object/communication manager **443**, and returns an error code if the object cannot be acquired.

Once the PTO **500** is acquired by object/communications manager **443**, object interpreter **435** begins to build PPT by parsing PTO **500** into its constituent segment calls to pages and page elements, as shown in FIG. **4***d* and interpreting such segments. PFO and PEO call segments **526** and **522** require the acquisition of the corresponding objects with object id's <ABCF>, <ABCX> and <ABCY>. Parsing and interpretation of object ABCY requires the further acquisition of program objects <ABCI> and <ABCJ>.

During the interpretation of the PEOs **504** for partitions **250** and **260**, other RS **400** events are triggered. This corresponds to transition (2) to interpret pre-processors state (C) in FIG. **6**. Presentation data **530** is sent to display manager **461** for display using a NAPLPS decoder within display manager **461**, and, as the PEO <ABCY> for partition **260** is parsed and interpreted by object interpreter **435**, parameters in program call segment **532** identify the program object <ABCI> as an initializer. Object interpreter **435** obtains the program object from object storage facility **439**, and makes a request to TBOL interpreter **438** to execute the initializer program object **508** <ABCI>. The initializer performs the operations specified above using facilities of the RS virtual machine. TBOL interpreter **438**, using operating environment **450**, executes initializer program object **506** <ABCI>, and may, if a further program object **508** is required in the execution of the initializer, make a synchronous application level object request to object storage facility **439**. When the initializer terminates, control is returned to object interpreter **435**, shown as the return path in transition (2) in FIG. **6**.

Having returned to the process object state (B), object processor **435** continues processing the objects associated with PTO <ABC**1**>. Object interpreter continues to construct the PPT, providing RS **400** with an environment for subsequent processing of the PTO <ABC**1**> by pre-processors and post-processors at the page, partition, and field levels. When the PPT has been constructed and the initializer executed, control is returned to keyboard manager **434**, and the RS enters the wait for event (E) State, via transition (4), as shown in FIG. **6**.

In the wait for event state, the partitioned application waits for the user to create an event. In any partitioned application, the user has many options. For example, the user may move the cursor to the "JUMP" field **296** on the command bar **290**, which is outside the current application, and thus cause subsequent navigation to another application. For purposes of this example, it is assumed that the user enters the number of apples he wishes to order by entering a digit in display field **271**.

Keyboard manager **434** translates the input from the user's keyboard to a logical representation independent of any type of personal computer. Keyboard manager. **434** saves the data entered by the user in a buffer associated with the current field defined by the location of the cursor. The

buffer is indexed by its PEV number, which is the same as the field number assigned to it during the formation of the page element. Keyboard manager **434** determines for each keystroke whether the keystroke corresponds to an input event or to an action or completion event. Input events are logical keystrokes and are sent by keyboard manager to display manager **461**, which displays the data at the input field location. Display manager **461** also has access to the field buffer as indexed by its PEV number.

The input data are available to TBOL interpreter **438** for subsequent processing. When the cursor is in a partition, only the PEVs for that partition are accessible to the RS virtual machine. After the input from the user is complete (as indicated by a user action such as pressing the RETURN key or entry of data into a field with an action attribute), RS **400** enters the Process Event state (E) via transition (4).

For purposes of this example, let us assume that the user enters the digit "5" in input field **270**. A transition is made to the process event state (E). Keyboard manager **434** and display manager **437** perform a number of actions, such as the display of the keystroke on the screen, the collection of the keystroke for input, and optionally, the validation of the keystroke, i.e. numeric input only in numeric fields. When the keystroke is processed, a return is made to the wait for event state (D) Edit attributes are specified in the field definition segment.

Suppose the user inputs a "6" next. A transition occurs to the PE state and after the "6" is processed, the Wait for Event (D) state is reentered. If the user hits the "completion" key (e.g., ENTER) the Process Event (E) state will be entered. The action attributes associated with field **272** identify this as a system event to trigger post-processor program object <ABCJ>. When the interpretive execution of program object <ABCJ> is complete, the wait for event state (D) will again be entered. The user is then free to enter another value in the input field, or select a command bar function and exit the apples application.

While this invention has been described in its preferred form, it will be appreciated that changes may be made in the form, construction, procedure and arrangement of its various elements and steps without departing from its spirit or scope.

We claim:

1. A method for presenting advertising obtained from a computer network, the network including a multiplicity of user reception systems at which respective users can request applications, from the network, that include interactive services, the respective reception systems including a monitor at which at least the visual portion of the applications can be presented as one or more screens of display, the method comprising the steps of:

a. structuring applications so that they may be presented, through the network, at a first portion of one or more screens of display; and

b. structuring advertising in a manner compatible to that of the applications so that it may be presented, through the network, at a second portion of one or more screens of display concurrently with applications, wherein structuring the advertising includes configuring the advertising as objects that include advertising data and;

c. selectively storing advertising objects at a store established at the reception system.

2. The method of claim **1** wherein storing advertising objects at the reception system includes replenishing the store of advertising objects from the network when the store of advertising objects falls below a predetermined level.

3. The method of claim **2** wherein storing advertising objects at the reception system includes storing advertising object identifications at the reception system, and wherein the storing of advertising object identification is based on an establishing of a characterization for the respective reception system users.

4. The method of claim **3** wherein establishing the characterization for the respective reception system users includes basing the characterization at least in part on the applications requested by the respective users.

5. The method of claim **3** wherein establishing the characterization for the respective reception system users includes basing the characterization at least in part on the demographic data for the respective users.

6. The method of claim **3** wherein establishing the characterization for the respective reception system users includes basing the characterization at least in part on data concerning the geographical location of the respective user's reception system.

7. The method of claim **3** wherein establishing the characterization for the respective reception system users includes basing the characterization at least in part on a combination of data concerning user application requests, user demographics and geographical location of the respective user's reception system.

8. A method for presenting advertising in a computer network, the network including a multiplicity of user reception systems at which respective users can request applications that include interactive services, the method comprising the steps of:

a. compiling data concerning the respective users;

b. establishing characterizations for respective users based on the compiled data; and

c. structuring advertising so that it may be selectively supplied to and retrieved at the reception systems for presentation to the respective users in accordance with the characterizations established for the respective reception system users, wherein structuring advertising includes supplying advertising data to the reception system and storing a predetermined amount of the advertising data in a store established at the respective reception systems.

9. The method of claim **8** wherein supplying advertising data to the reception system includes pre-fetching advertising data from the network when the store of advertising data falls below a predetermined level.

10. The method of claim **9** wherein pre-fetching advertising data is dependent on the size of the advertising data store.

11. The method of claim **10** wherein storing advertising data at the reception system includes maintaining a list identifying the advertising data to be presented.

12. The method of claim **8** wherein the supplying of advertising data to the reception system for presentation includes the reception system requesting advertising data from the network when advertising data sought to be presented is unavailable at the reception system.

13. A method for presenting advertising in a computer network, the network including a multiplicity of user reception systems at which respective users can request applications that include interactive services, the respective reception systems including a monitor at which at least the visual portion of the applications can be presented as one or more screens of display, the method comprising the steps of:

a. structuring applications so that they may be presented at a first portion of one or more screens of display;

b. configuring the advertising as objects that include advertising data,

c. structuring the advertising objects in a manner compatible to that of the applications so that advertising data from an advertising object may be presented at a second portion of one or more screens of display concurrently with applications, and;

d. selectively storing advertising objects at a store established at the reception system.

14. A method for presenting advertising obtained from a computer network, the network including a multiplicity of user reception systems at which respective users can request applications from the network that include interactive services, the respective reception systems including a monitor at which at least the visual portion of the applications can be presented as one or more screens of display, the method comprising the steps of:

a. structuring applications so that a user requested application may be presented, through the network, at a first portion of one or more screens of display;

b. separately structuring the advertising in a manner compatible to that of the applications so that advertising may be presented, through the network, at a second portion of one or more screens of display concurrently with any one of a plurality of user requested applications,

c. configuring the advertising as objects that include advertising data, and

d. selectively storing advertising objects at a store established at the reception system.

15. The method of claim **14** wherein storing advertising objects at the reception system includes replenishing the store of advertising objects from the network when the store of advertising objects falls below a predetermined level.

16. The method of claim **15** wherein storing advertising objects at the reception system includes storing advertising object identifications at the reception system, and wherein the storing of advertising object identification is based on an establishing of a characterization for the respective reception system users.

17. The method of claim **16** wherein establishing the characterization for the respective reception system users includes basing the characterization at least in part on the applications requested by the respective users.

18. The method of claim **16** wherein establishing the characterization for the respective reception system users includes basing the characterization at least in part on the demographic data for the respective users.

19. The method of claim **16** wherein establishing the characterization for the respective reception system users includes basing the characterization at least in part on data concerning the geographical location of the respective user's reception system.

20. The method of claim **16** wherein establishing the characterization for the respective reception system users includes basing the characterization at least in part on a combination of data concerning user application requests, user demographics and geographical location of the respective user's reception system.

21. A method for presenting advertising obtained from a computer network, the network including a multiplicity of user reception systems at which respective users can request, from the network, applications that include interactive services, the method comprising the steps of:

a compiling data concerning the respective users;

b. establishing characterizations for respective users based on the compiled data; and

c. structuring advertising separately from the applications so that the advertising may be selectively supplied, through the network, to and retrieved at the reception systems for presentation to the respective users along with a requested application in accordance with the characterizations established for the respective reception system users,

wherein supplying advertising data to the reception system includes storing a predetermined amount of the advertising data in a store established at the respective reception systems.

22. The method of claim **21** wherein supplying advertising data to the reception system includes pre-fetching advertising data from the network when the store of advertising data falls below a predetermined level.

23. The method of claim **22** wherein pre-fetching advertising data is dependent on the size of the advertising data store.

24. The method of claim **23** wherein storing advertising data at the reception system includes maintaining a list identifying the advertising data to be presented.

25. The method of claim **21** wherein the supplying of advertising data to the reception system for presentation includes the reception system requesting advertising data from the network when advertising data sought to be presented is unavailable at the reception system.

* * * * *

EXHIBIT C

US005961601A

United States Patent [19]

Iyengar

[11] Patent Number: 5,961,601

[45] Date of Patent: Oct. 5, 1999

[54] **PRESERVING STATE INFORMATION IN A CONTINUING CONVERSATION BETWEEN A CLIENT AND SERVER NETWORKED VIA A STATELESS PROTOCOL**

[75] Inventor: **Arun K. Iyengar**, Yorktown Heights, N.Y.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **08/660,633**

[22] Filed: **Jun. 7, 1996**

[51] **Int. Cl.**[6] **G06F 13/38**; G06F 15/17

[52] **U.S. Cl.** **709/229**; 709/228; 709/218; 709/203

[58] **Field of Search** 395/200.32, 200.48, 395/200.53, 200.59, 182.02, 182.03, 182.05; 709/202, 218, 223, 229, 228, 203; 714/4, 5, 7

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,218,695	6/1993	Noveck et al. .	
5,623,656	4/1997	Lyons	395/200.49
5,668,943	9/1997	Attanasio et al.	395/182.05
5,701,451	12/1997	Rogers et al.	395/200.32
5,708,780	1/1998	Levergood et al.	395/200.12
5,710,918	1/1998	Lagarde et al.	395/200.32
5,774,670	6/1998	Montulli	395/200.57

FOREIGN PATENT DOCUMENTS

0604010	10/1993	European Pat. Off. .
0625750	11/1994	European Pat. Off. .

OTHER PUBLICATIONS

"Haht Software Premiers Hahtsite at Demo 96", PR Newswire, Jan. 29, 1996.

"Proposed HTTP State Management Mechanism", Montulli et al., HTTP Working Group, Feb. 16, 1996.

Louis Perrochon et al., "IDLE: Unified W3–access to interactive information servers", Computer Networks and ISDN Systems, Elsevier Science B.V., pp. 927–938, (1995).

Bertrand Ibrahim, "World–wide algorithm animation", Computer Networks and ISDN Systems, Elsevier Science B.V., pp. 255–265, (1994).

Alan Falconer Slater, "Extending W3 clients", Computer Networks and ISDN Systems, Elsevier Science B.V., pp. 61–68, (1995).

(List continued on next page.)

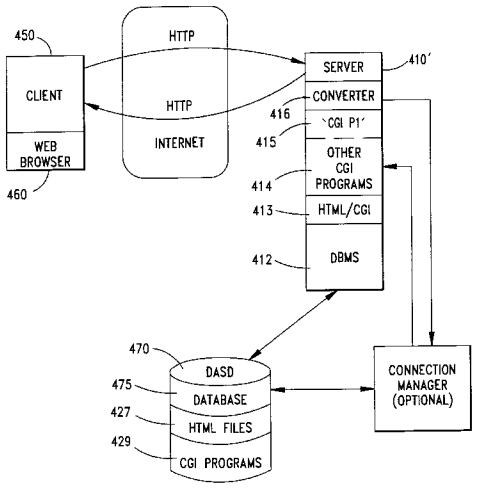
Primary Examiner—Mark H. Rinehart

Attorney, Agent, or Firm—Kevin M. Jordan

[57] **ABSTRACT**

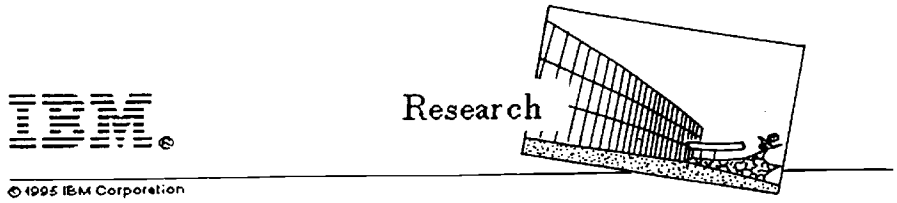
A method and system for preserving state in computers communicating over networks, such as the World Wide Web (WWW) using stateless protocols, e.g., HTTP. State is preserved in a conversation between a client requesting services from a served by performing the service and identifying all continuations (hyperlinks) in an output from the service; recursively embedding the state information in all identified continuations in the output sent to the client. The state information may be embedded and communicated by the server to the client. Alternatively, dynamically downloadable program code may be used to embed the state information at the client. Additional features enable the filtering and/or addition of hyperlinks and data output from the services according to predetermined criteria. State information may be embedded by modifying an identified continuation which is a request for an HTML file, to invoke a CGI converter program with the identified continuation and the state information passed as arguments. State information may also be embedded by modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, and the embedding step is performed by the CGI program. Alternatively, an identified continuation which is an invocation of a CGI program may be modified to invoke a CGI converter program with the identified continuation, an argument counter which indicates a number of arguments associated with the CGI program, and the state information passed as arguments. Here, the embedding is performed by the converter program.

68 Claims, 10 Drawing Sheets

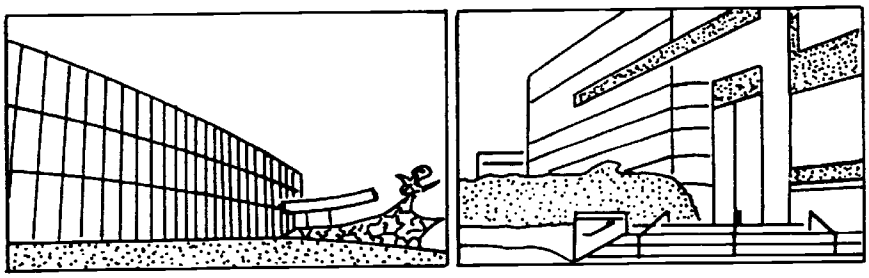


OTHER PUBLICATIONS

"Behind The Scenes of The Adventure Web", http://tjww-w.stanford.edu/adventure/impl.html Feb. 23, 1996.
"Aug. 1995 Web Watch", http:www.netgen.com/corpinfo/press/webwtchv6n8.html Feb. 2, 1996.
"Zebrafish Database", http://zfish.uoregon.edu/zf info/dbase/arch.html Oct. 24, 1995.
"Porting Interactive Applications to the Web", http://ksi.cp-sc.ucalgary.ca/articles/WWW/PortWeb/PortWeb.html Dec. 7, 1995.
"MBA/MBARI Live Link to The Technology Museum of Innovation in San Jose" http://rockfish.mbari.org/BayLink/http prb.html Aug. 2, 1995.
"Persistent Client State HTTP Cookies", Netscape Communications Corporation, 1996 http://home.netscape.com/newsref/std/cookie_spec.html.
"Hypertext Transfer Protocol—HTTP/1.0" http://www.ic-s.uci.edu/pub/ietf/http/draft–ietf–http–v10–spec–03.html, by T. Berners–Lee, R. Fielding, and H. Frystyk, Sep. 4, 1995.
"Network News Transfer Protocol: A Proposed Standard for the Stream–Based Transmission of News", RFC 977, B. Kantor and P. Lapsley, UC San Diego and UC Berkeley, Feb. 1986, http://ds.internic.net/rfc/rfc977.txt.
"Simple Mail Transfer Protocol", RFC 821, J.B. Postel, Information Sciences http://ds.internic.net/std/std10.txt.
J. Postel and J.K. Reynolds, "File Transfer Protocol (FTP)", RFC 959, Information Sciences Institute, USC, Oct. 1985 http://ds.internic.net/std/std9.txt.



IBM T.J. Watson Research Center



T.J. Watson Research Center: Yorktown (left) and Hawthorne.

- Welcome!
- Local Education Outreach
- Visitor info and local site directions
- Local hotels
- IBM home page -- IBM Research home page

Click on icon to send your comments.

Or, contact *webmaster@watson.ibm.com*

[IBM home page | Order | Search | Contact IBM | Help | (C) | (TM)]

FIG. 1
PRIOR ART

FIG.2

PRIOR ART

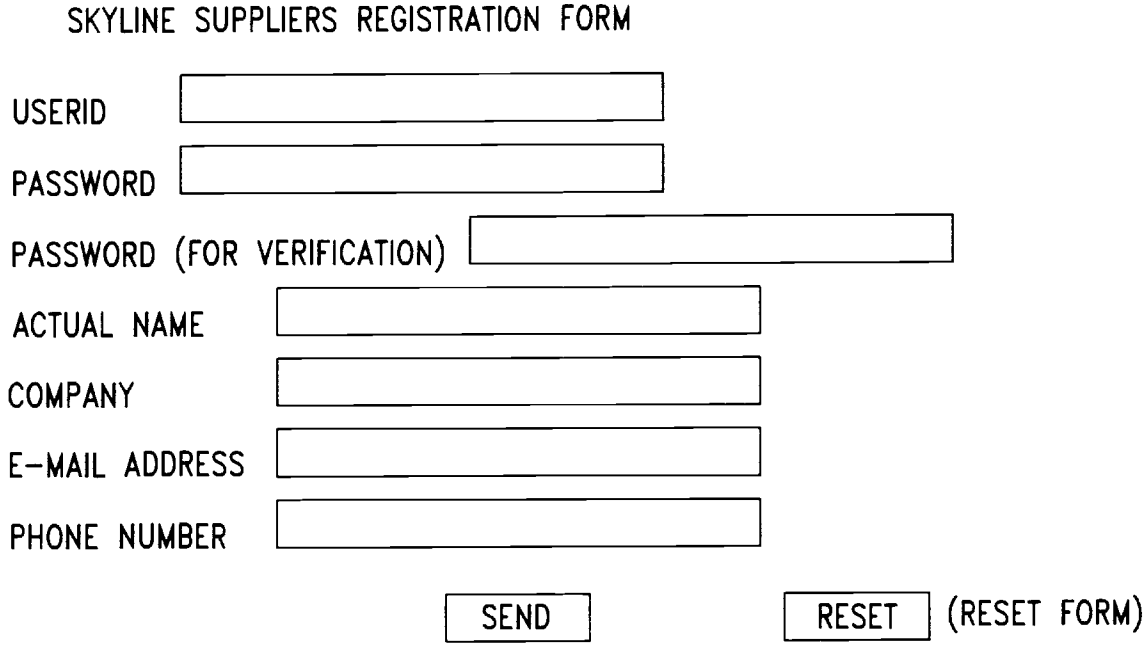


FIG.3

PRIOR ART

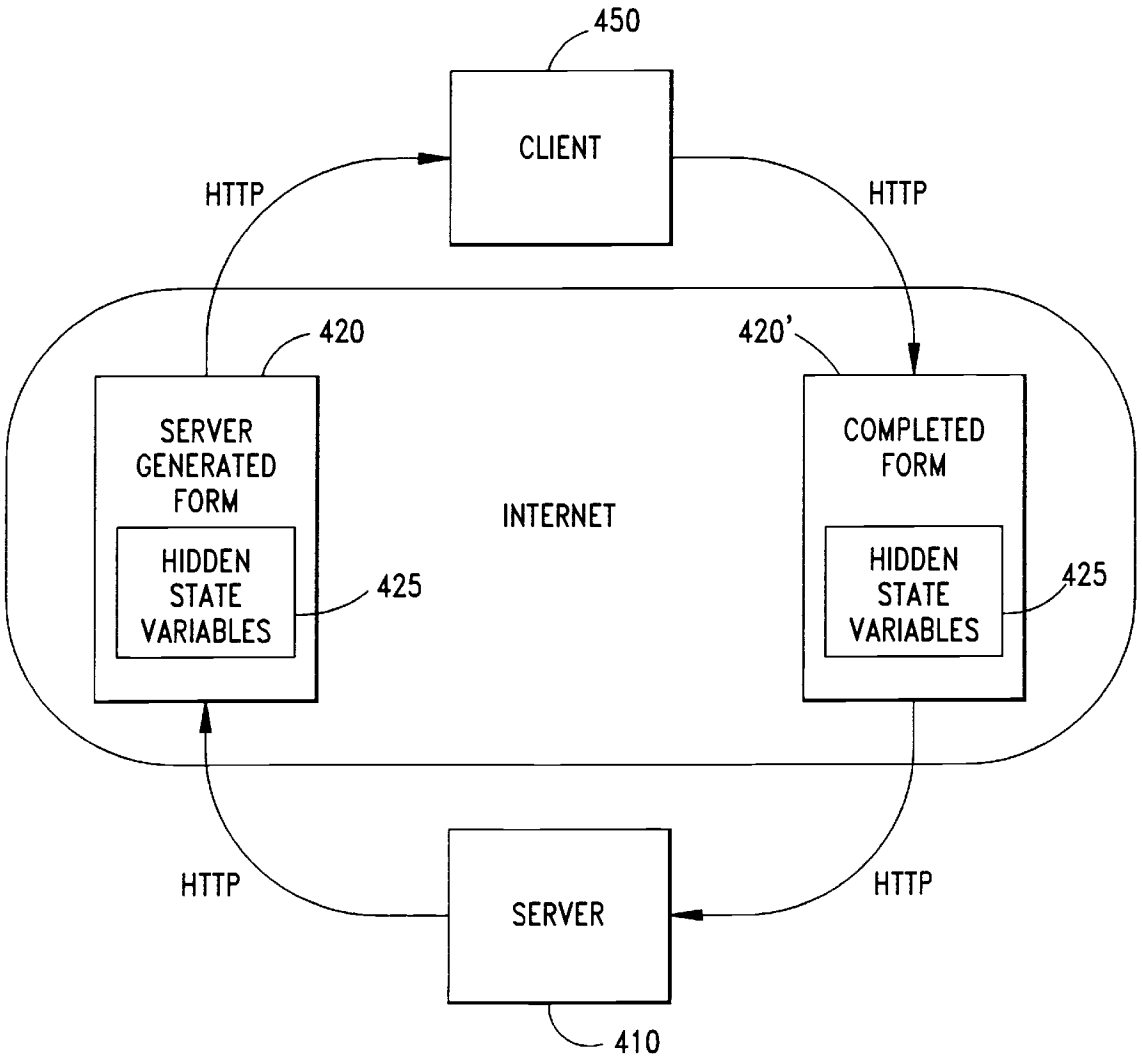
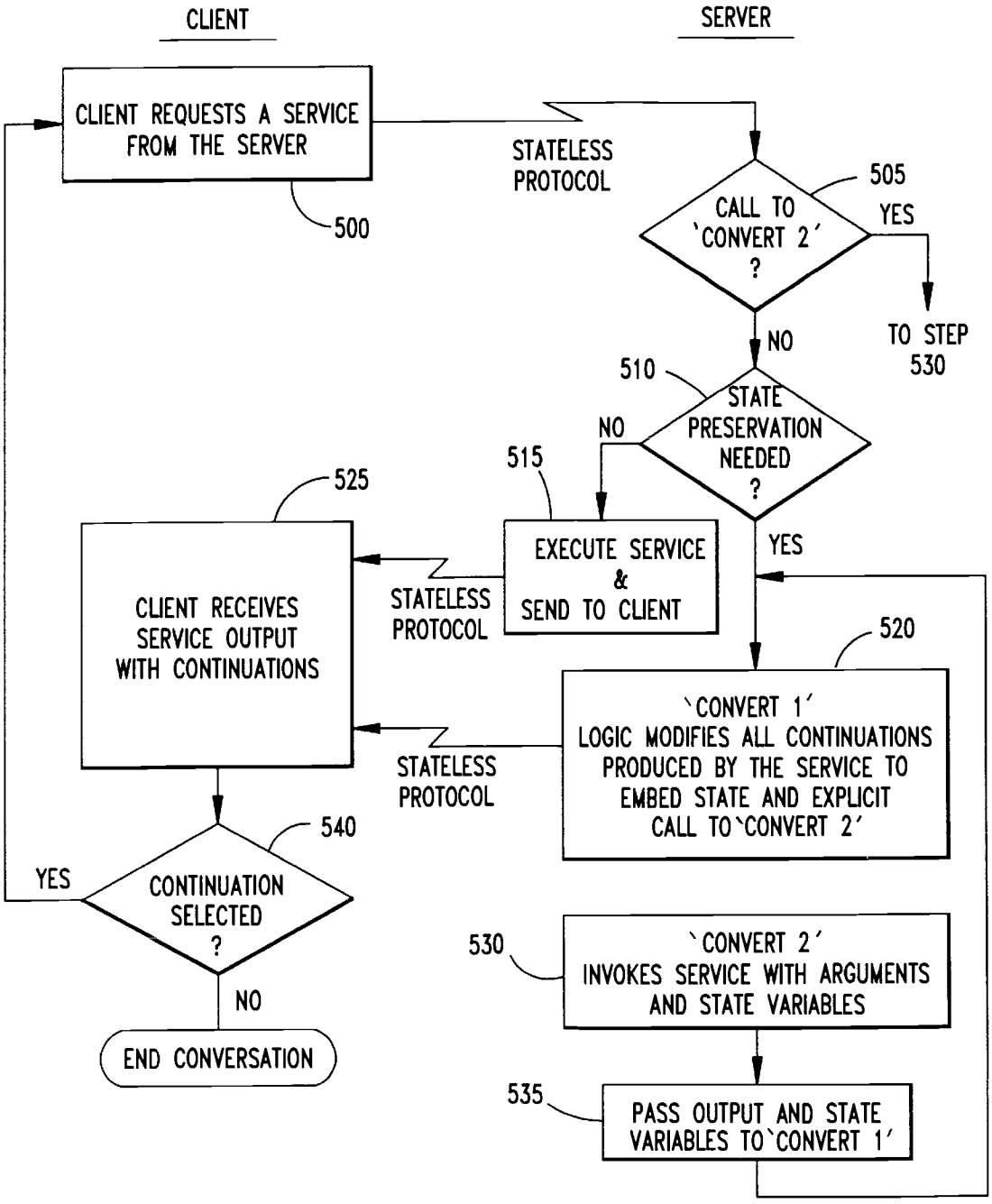


FIG.4


CLIENT
SERVER
CLIENT REQUESTS A SERVICE FROM THE SERVER
500
STATELESS PROTOCOL
505
CALL TO `CONVERT 2' ?
YES
TO STEP 530
NO
510
STATE PRESERVATION NEEDED ?
NO
YES
515
EXECUTE SERVICE & SEND TO CLIENT
STATELESS PROTOCOL
525
CLIENT RECEIVES SERVICE OUTPUT WITH CONTINUATIONS
520
`CONVERT 1' LOGIC MODIFIES ALL CONTINUATIONS PRODUCED BY THE SERVICE TO EMBED STATE AND EXPLICIT CALL TO `CONVERT 2'
STATELESS PROTOCOL
540
CONTINUATION SELECTED ?
YES
NO
END CONVERSATION
530
`CONVERT 2' INVOKES SERVICE WITH ARGUMENTS AND STATE VARIABLES
535
PASS OUTPUT AND STATE VARIABLES TO `CONVERT 1'

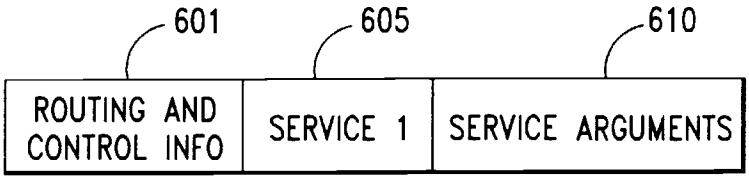

601
605
610
ROUTING AND CONTROL INFO
SERVICE 1
SERVICE ARGUMENTS

FIG.5

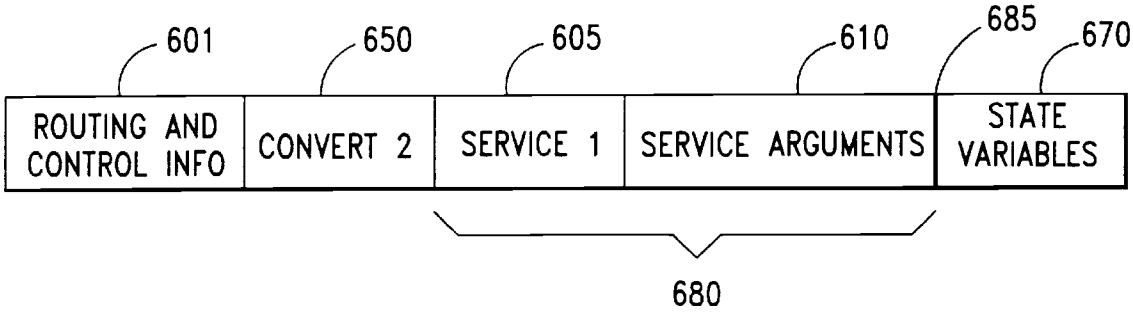

601
650
605
610
685
670
ROUTING AND CONTROL INFO
CONVERT 2
SERVICE 1
SERVICE ARGUMENTS
STATE VARIABLES
680

FIG.6

FIG.7a

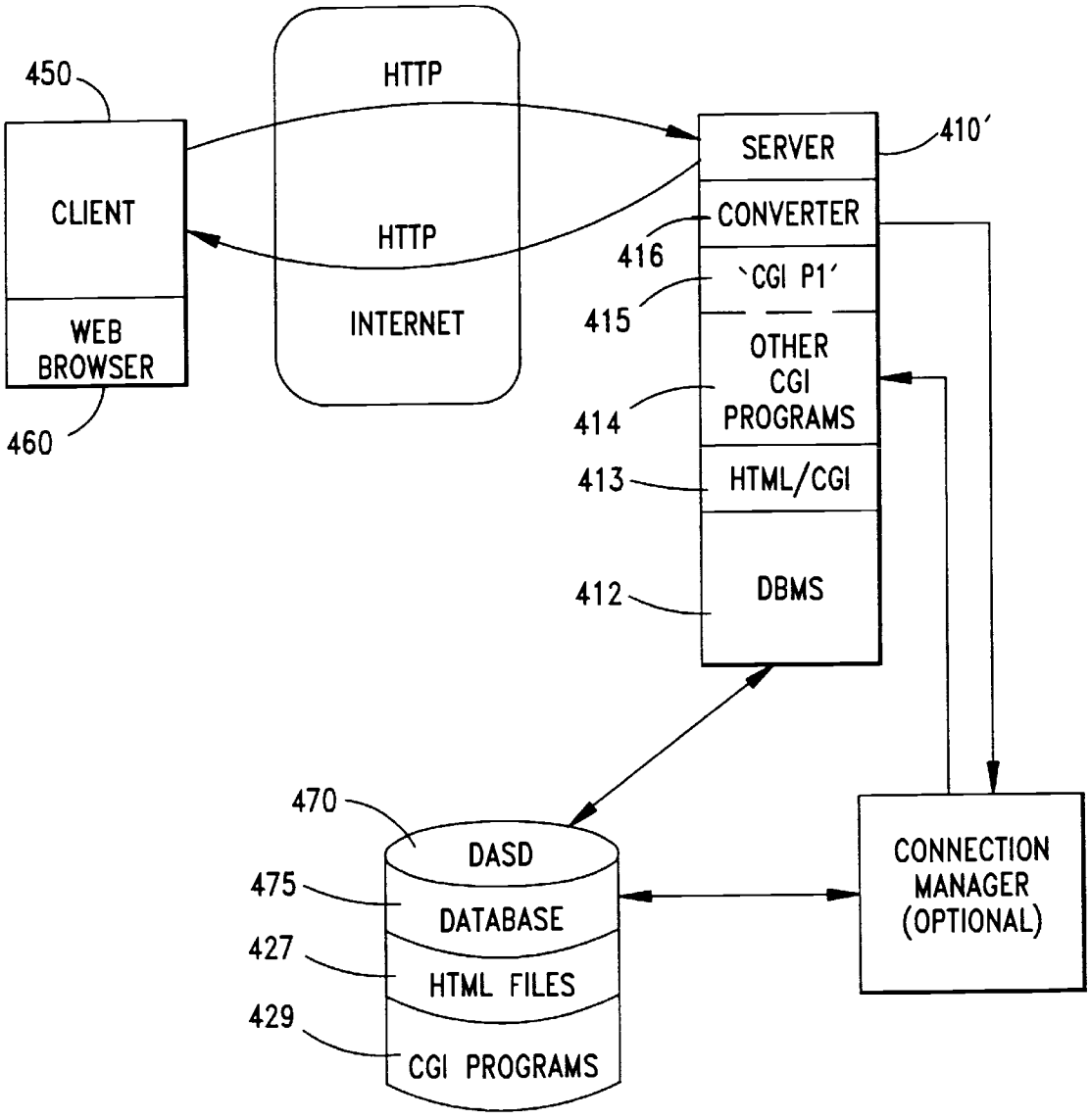

450
CLIENT
WEB BROWSER
460
HTTP
HTTP
INTERNET
SERVER
410'
CONVERTER
416
'CGI P1'
415
OTHER CGI PROGRAMS
414
HTML/CGI
413
DBMS
412
470
DASD
475
DATABASE
427
HTML FILES
429
CGI PROGRAMS
CONNECTION MANAGER (OPTIONAL)

FIG.7b

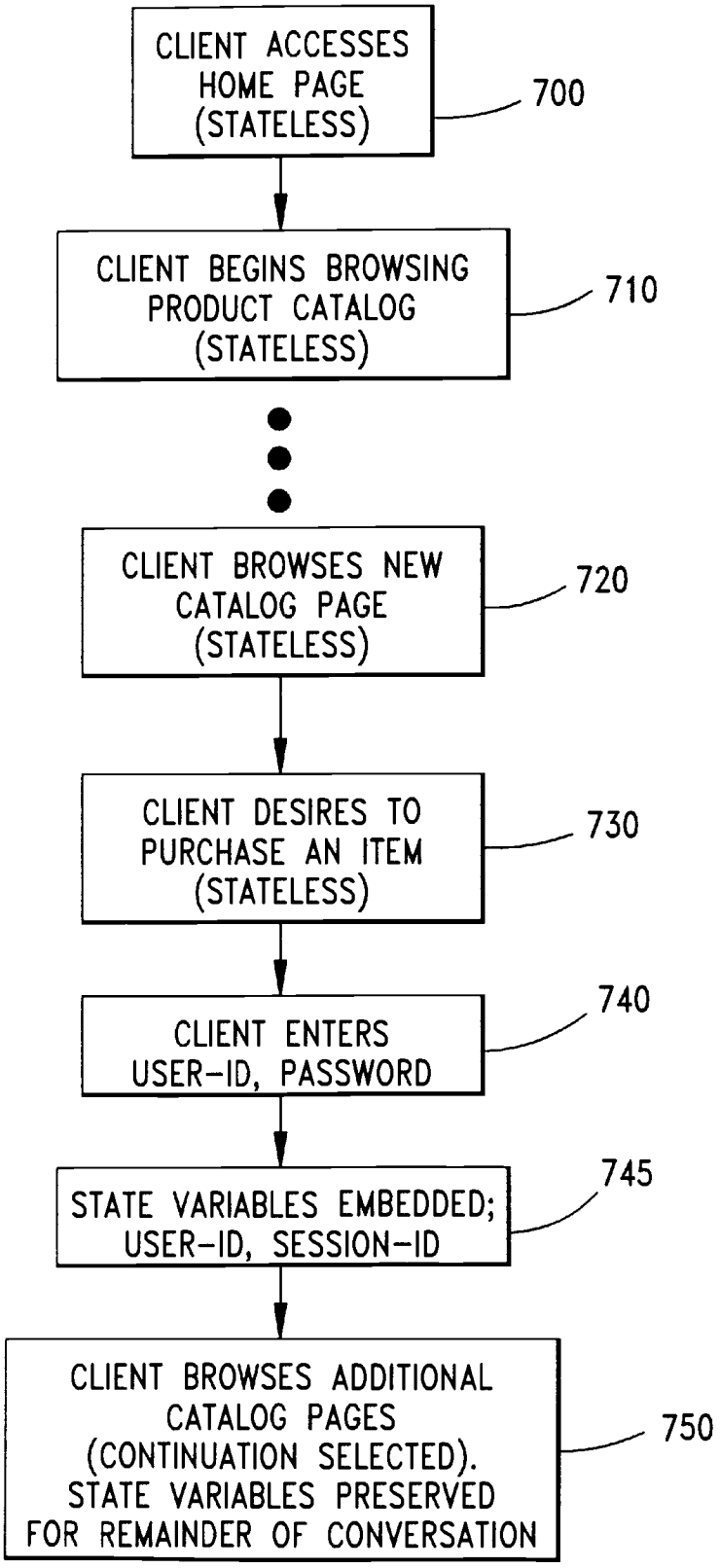
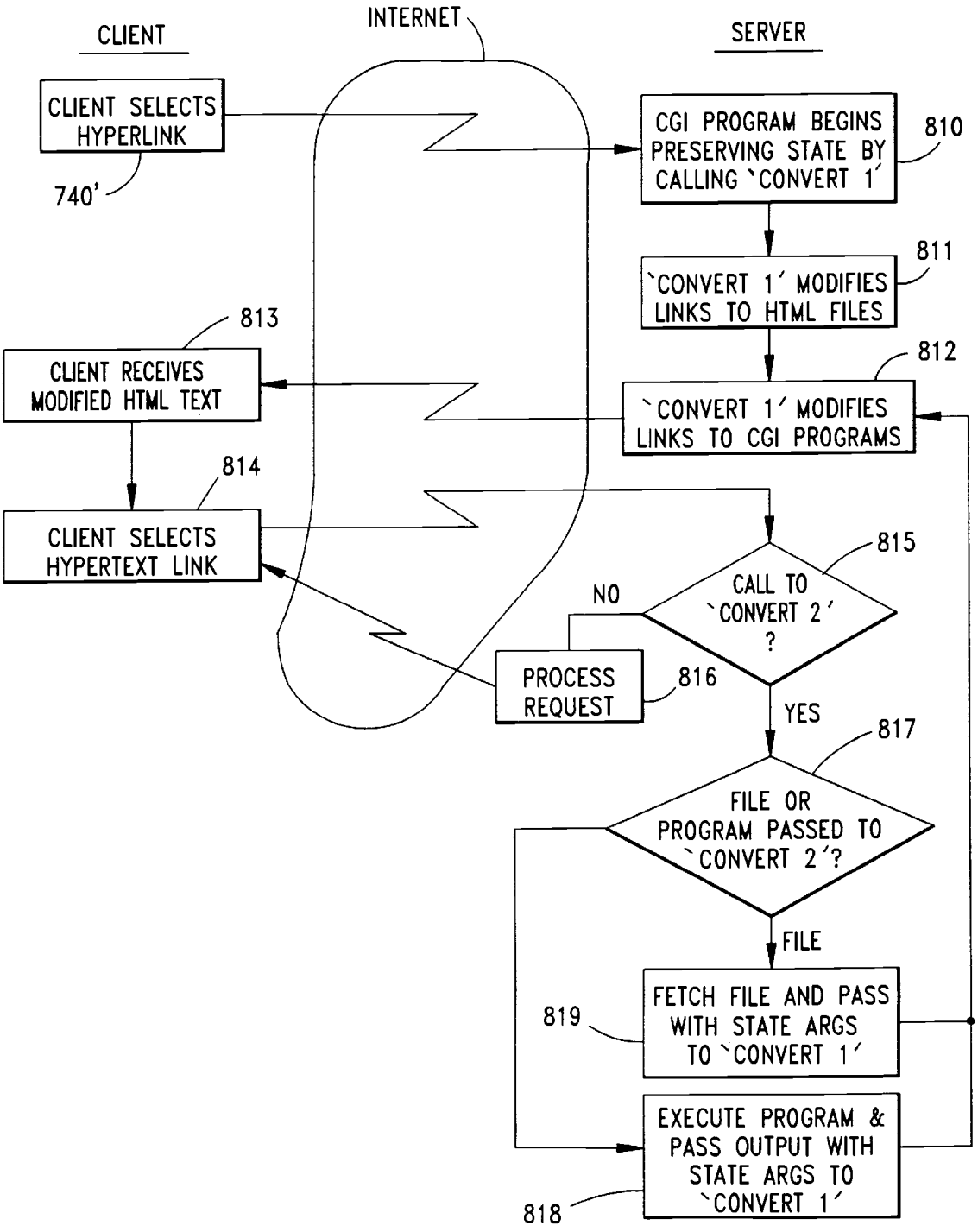
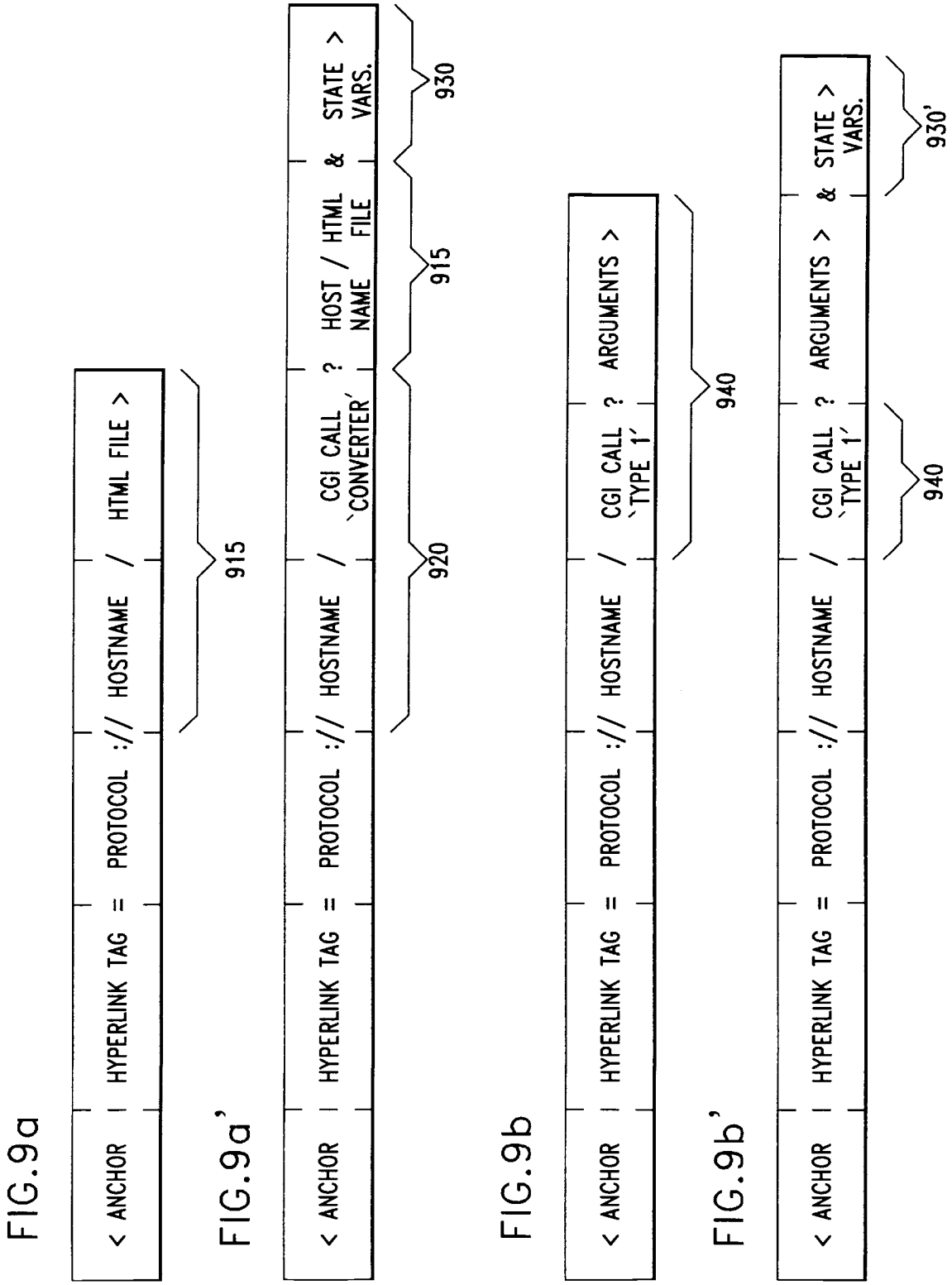

CLIENT ACCESSES HOME PAGE (STATELESS)
700
CLIENT BEGINS BROWSING PRODUCT CATALOG (STATELESS)
710
CLIENT BROWSES NEW CATALOG PAGE (STATELESS)
720
CLIENT DESIRES TO PURCHASE AN ITEM (STATELESS)
730
CLIENT ENTERS USER-ID, PASSWORD
740
STATE VARIABLES EMBEDDED; USER-ID, SESSION-ID
745
CLIENT BROWSES ADDITIONAL CATALOG PAGES (CONTINUATION SELECTED). STATE VARIABLES PRESERVED FOR REMAINDER OF CONVERSATION
750

FIG.8


CLIENT
INTERNET
SERVER
CLIENT SELECTS HYPERLINK
740'
CGI PROGRAM BEGINS PRESERVING STATE BY CALLING `CONVERT 1'
810
`CONVERT 1' MODIFIES LINKS TO HTML FILES
811
`CONVERT 1' MODIFIES LINKS TO CGI PROGRAMS
812
CLIENT RECEIVES MODIFIED HTML TEXT
813
CLIENT SELECTS HYPERTEXT LINK
814
CALL TO `CONVERT 2' ?
815
NO
PROCESS REQUEST
816
YES
FILE OR PROGRAM PASSED TO `CONVERT 2'?
817
FILE
FETCH FILE AND PASS WITH STATE ARGS TO `CONVERT 1'
819
EXECUTE PROGRAM & PASS OUTPUT WITH STATE ARGS TO `CONVERT 1'
818


< ANCHOR | HYPERLINK TAG = PROTOCOL :// HOSTNAME / HTML FILE >
915
FIG.9a
< ANCHOR | HYPERLINK TAG = PROTOCOL :// HOSTNAME / CGI CALL ? `CONVERTER' HOST / HTML NAME FILE & STATE > VARS.
920
915
930
FIG.9a'
< ANCHOR | HYPERLINK TAG = PROTOCOL :// HOSTNAME / CGI CALL ? `TYPE 1' ARGUMENTS >
940
FIG.9b
< ANCHOR | HYPERLINK TAG = PROTOCOL :// HOSTNAME / CGI CALL ? `TYPE 1' ARGUMENTS > & STATE > VARS.
940
930'
FIG.9b'

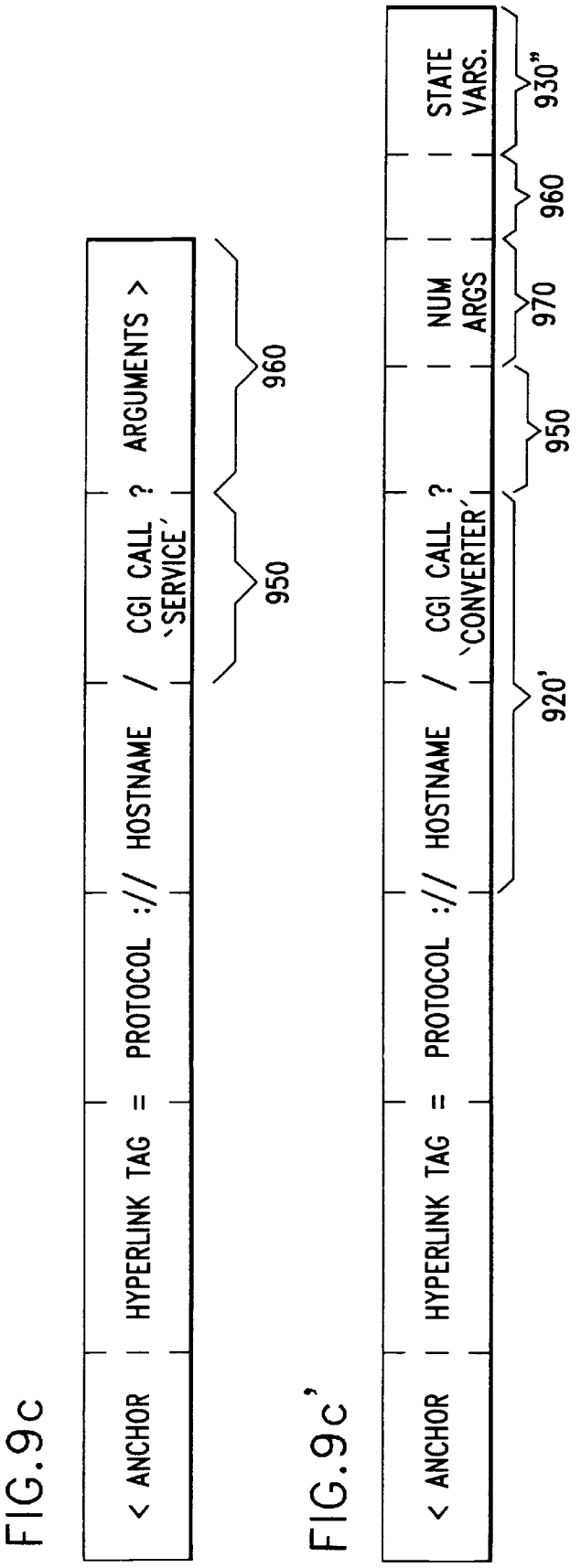

< ANCHOR | HYPERLINK TAG = PROTOCOL :// HOSTNAME / CGI CALL ? `SERVICE' ARGUMENTS >
950
960
< ANCHOR | HYPERLINK TAG = PROTOCOL :// HOSTNAME / CGI CALL ? `CONVERTER' | NUM ARGS | STATE VARS.
920'
950
970
960
930"

FIG.9c

FIG.9c'

PRESERVING STATE INFORMATION IN A CONTINUING CONVERSATION BETWEEN A CLIENT AND SERVER NETWORKED VIA A STATELESS PROTOCOL

FIELD OF THE INVENTION

This invention is related to computers and computer networks. In particular, the invention is related to computers preserving state while communicating over networks via stateless protocols. Even more particularly, the invention is related to a method and system for preserving state in computers communicating over the Internet, specifically the World Wide Web, using the HyperText Transfer Protocol (HTTP).

CROSS-REFERENCE TO RELATED PATENTS

The present invention is related to the following United States of America Patents:

U.S. Pat. No. 5,752,022, issued May 12, 1998, entitled: "A Method for Creating a Hypertext Language for a Distributed Computer Network," by Chiu et al.; and

U.S. Pat. No. 5,710,918, issued Jan. 20, 1998, entitled "Method for Distributed Task Fulfillment of Web Browser Requests," by Lagarde et al. These patents, which have a common assignee, International Business Machines Corporation, Armonk, N.Y., are hereby incorporated by reference in their entirety.

GLOSSARY OF TERMS

While dictionary meanings are also implied by certain terms used here, the following glossary of some terms may be useful.

Internet

The network of networks and gateways that use the TCP/IP suite of protocols.

TCP/IP

Transmission Control Protocol/Internet protocol. A packet switching scheme the Internet uses to chop, route, and reconstruct the data it handles, from e-mail to video.

Client

A client is a computer which issues commands to the server which performs the task associated with the command.

Server

Any computer that performs a task at the command of another computer is a server. A Web server typically supports one or more clients.

World Wide Web (WWW or Web)

The Internet's application that lets people seeking information on the Internet switch from server to server and database to database by clicking on highlighted words or phrases of interest (hyperlinks). An Internet WWW server supports clients and provides information. The Web can be considered as the Internet with all of the resources addressed as URLs and which uses HTML to display the information corresponding to URLs and provide a point-and-click interface to other URLs.

Universal Resource Locator (URL)

A way to uniquely identify or address information on the Internet. Can be considered to be a Web document version of an e-mail address. URLs can be cumbersome if they belong to documents buried deep within others. They can be accessed with a Hyperlink. An example of a URL is "http://www.arun.com:80/table.html". A URL has four components. Starting from the left, the first specifies the protocol to use, separated from the rest of the locator by a ":". Next is the hostname or IP address of the target host; this is delimited by the "//" on the left and on the right by a "/" or optionally a ":". The port number is optional, and is delimited on the left from the hostname by a ":" and on the right by a "/". The fourth component is the actual file name or program name. In this example, the ".html" extension means that this is an HTML file.

Hyperlink (or Hypertext Link)

A network address embedded in a word, phrase, icon or picture that is activated when you select it. Information about that item is returned to the client and displayed using a Web browser.

HyperText Markup Language (HTML)

HTML is the language used by Web servers to create and connect documents that are viewed by Web clients. HTML uses Hypertext documents. Other uses of Hypertext documents are described in U.S. Pat. No. 5,204,947, granted Apr. 20, 1993 to Bernstein et al.; U.S. Pat. No. 5,297,249, granted Mar. 22, 1994 to Bernstein et al.; U.S. Pat. No. 5,355,472, granted Oct. 11, 1994 to Lewis; all of which are assigned to International Business Machines Corporation, and which are incorporated by reference herein.

Hypertext Transfer Protocol (HTTP)

HTTP is an example of a stateless protocol, which means that every request from a client to a server is treated independently. The server has no record of previous connections. At the beginning of a URL, "http:" indicates the file contains hyperlinks.

Home Page

A multi-media table of contents that guides a web user to stored information, e.g., about an organization, on the Internet.

Web Browser

A program running on a computer that acts as an Internet tour guide, complete with pictorial desktops, directories and search tools used when a user "surfs" the Internet. In this application the Web browser is a client service which communicates with the World Wide Web.

HTTP Daemon (HTTPD)

An IBM OS/2 Web Server or other server having Hypertext Markup Language and Common Gateway Interface capability. The HTTPD is typically supported by an access agent which provides the hardware connections to machines on the intranet and access to the Internet, such as TCP/IP couplings.

Continuations

Hypertext links (or hyperlinks) are examples of continuations in client-server communications. A continuation is a new request which a client may send to a server. Whenever a client requests something from a server, the server may include one or more continuations in its response. When a server responds to a request, it may include one or more continuations which could be any valid requests. However, useful continuations are generally logically related to the original request.

Conversation

A sequence of communications between a client and server in which the server responds to each request with a set of continuations and the client always picks the next request from the set of continuations. On the Web, hypertext links represent continuations and a client engages in a conversation whenever it follows hypertext links.

BACKGROUND

Networks have transformed the way people do computing. Someone with access to a personal computer or work-

station can connect to the Internet and communicate with systems and people all over the world. The World Wide Web (WWW or Web) is a way of using the Internet that provides the user with access via linked documents to a wealth of information distributed throughout the globe. The WWW also allows users to execute programs running on remote servers. This capability enables users to obtain the results from programs which the user cannot run locally due to hardware and/or software limitations. It is also possible to download and run programs stored remotely on the World Wide Web. This has the potential to greatly increase the amount of software which is available to a computer connected to the World Wide Web.

Network Protocols

Network protocols provide standard methods for machines to communicate with one another. The protocols indicate how data should be formatted for receipt and transmission across networks. Heterogeneous machines can communicate seamlessly over a network via standard protocols. Examples of standard Internet protocols include: HTTP, see, e.g., "Hypertext Transfer Protocol—HTTP/1.0", http://www.ics.uci.edu/pub/ietf/http/draft-ietf-http-v10-spec-03.html, by T. Berners-Lee, R. Fielding, and H. Frystyk, Sep. 4, 1995; SMTP, see, e.g, "Simple Mail Transfer Protocol". RFC 821, J. B. Postel, Information Sciences Institute, USC, August 1982, http://ds.internic.net/std/std10.txt.; NNTP, see, e.g., "Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News", RFC 977, B. Kantor and P. Lapsley, UC San Diego and UC Berkeley, February 1986, http://ds.internic.net/rfc/rfc977.txt.; FTP, see e.g., J. Postel and J. K. Reynolds. "File Transfer Protocol (FTP)", RFC 959, Information Sciences Institute, USC, October 1985, http://ds.internic.net/std/std9.txt.; Gopher, see, e.g., F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti. "The Internet Gopher Protocol: A distributed document search and retrieval protocol", RFC 1436, University of Minnesota, March 1993, http://ds.internic.net/rfc/rfc1436.txt.; and WAIS, see, e.g., F. Davis, B. Kahle, H. Morris, J. Salem, T. Shen, R. Wang, J. Sui, and M. Grinbaum. "WAIS Interface Protocol Prototype Functional Specification" (v 1.5), Thinking Machines Corporation, April 1990.

The client-server model constitutes one of the dominant paradigms in network programming, see, e.g., W. R. Stevens, "Unix Network Programming", Prentice Hall PTR, Englewood Cliffs, N.J., 1990; and D. E. Comer, "Internetworking with TCP/IP" vol 1., Prentice Hall, Englewood Cliffs, N.J., 1991 which is hereby incorporated by reference in its entirety. A server program offers a service which can be accessed by multiple users over the network. A program becomes a client when it sends a message to a server and waits for a response from the server. The client process, which is typically optimized for user interaction, uses the requested service without having to know any of the detailed workings of the requested service or server. On the World Wide Web, "browsers" constitute client programs while the programs sending back information to the browser constitute server programs.

A client and server may communicate either synchronously or asynchronously. In a synchronous communication, a client waits for a response from a server before issuing the next request. In an asynchronous communication, the client may issue a request to a server before one or more responses from previous requests to the server have been received.

Many network protocols between a client and server are stateless. This means that every request from a client to a server is treated independently. The server has no record of previous connections. HTTP is an example of a stateless protocol. Two advantages of using stateless protocols are efficiency and simplicity. However, there are situations where it is desirable for maintaining state information during communications between the client and server. For these types of interactions, the statelessness of protocols can present problems.

The HTTP Protocol and the World Wide Web

The most compelling application of the present invention is for browsing the World Wide Web via the HTTP protocol, see, e.g., "Hypertext Transfer Protocol—HTTP/1.0", http://www.ics.uci.edu/pub/ietf/http/draft-ietf-http-v10-spec-03.html, by T. Berners-Lee, R. Fielding, and H. Frystyk, Sep. 4, 1995, which is hereby incorporated by reference in its entirety. Those skilled in the art will understand, however, that the present invention is not limited to HTTP. The relevant aspects of the Web and the limitations imposed by the statelessness of protocols, such as HTTP, will now be discussed.

The World Wide Web consists of multiple servers networked together. Clients typically communicate with servers using a standard browser such as are sold under the trademarks "NETSCAPE NAVIGATOR" by Netscape, "MOSAIC" from NCSA, or "WEB EXPLORER" by IBM. The most common method of communicating between clients and servers is via the HTTP protocol. HTTP allows the client to obtain data from the server either by requesting a file or invoking a program known as a Common Gateway Interface (CGI) program which executes on the server. CGI programming is well known in the art. See, e.g.,"HTML and CGI Unleashed" by John December and Mark Ginsburg, Sams.net Publishing, Indianapolis, Ind. (1995). The server then sends file or the output from the CGI program to the client. Servers typically restrict the files and programs which a client has the ability to access.

The server sends information to the client using the HyperText Markup Language (HTML), see, e.g., "The HTML Sourcebook" by Ian S. Graham, John Wiley & Sons, Inc., New York, 1995, which is hereby incorporated by reference in its entirety. HTML documents consist of conventional ASCII text in which the information to be displayed is interspersed with HTML markup tags. These tags are surrounded by greater than and less than signs (< . . . >) and instruct the browser how to interpret different parts of documents. Browsers use Uniform Resource Locators (URLs) to uniquely identify or address information on the Internet. Browsers read HTML documents corresponding to the URLs and display them by following the instructions stored in the markup tags.

The HTML code sequence below (Table 1) shows the HTML text corresponding to the Web home page of the IBM T. J. Watson Research Center on Jun. 3, 1996. This Web page corresponds to the URL "http://www.watson.ibm.com/". The corresponding output that would be displayed on a standard browser accessing this page is shown in FIG. **1**.

TABLE 1

The HTML source code corresponding to the IBM T. J Watson Research Center home page.

```
<HTML><HEAD>
<TITLE>IBM T. J. Watson Research Center home page</TITLE>
<meta name="owner" content="calyson@watson.ibm.com">
<meta name="review" content="19960202">
</HEAD>
<BODY>
<IMG SRC="/watson/mast.gif" alt="Research" >
<p>
<h1>IBM T.J. Watson Research Center</h1>
<p>
<IMG SRC="/watson/night.gif" > <IMG SRC="/watson/haw2.gif" >
<br>
<i>T.J. Watson Research Center: Yorktown (left) and Hawthorne.</i>
<p>
<ul>
<IMG align=middle SRC="/watson/bullet.gif" ><A HREF="/watwel.html" >
Welcome!</a>
<br>
<IMG align=middle SRC="/watson/bullet.gif" ><A HREF="/leo" >Local Education Outreach
</a>
<br>
<IMG align=middle SRC="/watson/bullet.gif" ><A HREF="/menu.html" > Visitor info and local
site directions </a>
<br>
<IMG align=middle SRC="/watson/bullet.gif" ><A HREF="/lodging.html" > Local hotels</a>
<br>
<IMG align=middle SRC="/watson/bullet.gif" ><A href="http://www.ibm.com"> IBM home
page</a> -- <A href="http://www.research.ibm.com/"> IBM Research home page</a>
<br>
<ul>
<p>
<hr>
<A HREF="/watson/mail.html" ><IMG align=middle
SRC="/research/images/mail.gif" ></a> <b>Click on icon to send your comments.</b>
<p>
Or, contact <i>webmaster@watson.ibm.com</i>
<p>
<hr>
<Address><homepage@watson.ibm.com></address>
<b>
[
<A href="http://www.ibm.com/">IBM home page</a>|
<A href="http://www.ibm.com/Orders/">Order</a>|
<A href="http://www.austin.ibm.com/search/">Search</a>|
<A href="http://www.ibm.com/Assist/">Contact IBM</a>|
<A href="http://www.ibm.com/Finding/">Help</a>|
<A href="http://www.ibm.com/copyright.html">(C)</a>|
<A href="http://www.ibm.com/trademarks.html">(TM)</a>
]
</b>

</BODY>
</HTML>
```

Many Web browsers allow users to view the HTML source code of any document being viewed. The HTML text in Table 1 is stored in a file accessible to a Web server at the IBM T. J. Watson Research Center. When this Web server receives a request for the URL "http://www.watson.ibm.com/", it sends the appropriate file to the client's browser. The client's browser will then read and display the HTML file. (Table 1 contains a number of relative links. The hypertext links and image files are only valid if the file is stored in a specific directory. If, for example the "night.gif" file in Table 1 is stored at an arbitrary location, the hypertext links will be invalid and the associated images will not appear.)

The line in Table 1 reading "Visitor info and local site directions" is an example of a hypertext link (also called a hyperlink). The corresponding output as it would be displayed by a standard browser is depicted in FIG. **1**. When the user clicks on this link as depicted in FIG. **1** when displayed by the browser, a new HTML file, "menu.html", is fetched from the server and displayed by the browser. Hypertext links to documents on both local and remote servers can be placed in an HTML file. The ability to incorporate hyperlinks within an HTML file to link documents on servers all over the world is one of the key features of the World Wide Web. In other words, a Web browser can be used to access information from servers all over the world by simply pointing and clicking on hypertext links.

Recall that Hypertext links are examples of "continuations" in client-server communication. A continuation is a new request which a client may send to a server. Whenever a client requests something from a server, the server may include one or more continuations in its response. The continuations could represent any valid requests. However, useful continuations are generally logically related to the original request. A good set of continuations makes it easy for a client to communicate synchronously with a server.

After each request, the server responds with a set of continuations. The client chooses one of the continuations for the next request. A "conversation" is a sequence of communications between a client and server in which the server responds to each request with a set of continuations and the client always picks the next request from the set of continuations.

On the Web, hypertext links represent continuations and a client engages in a conversation whenever it follows hypertext links. A conversation is interrupted whenever the client obtains a new page by explicitly requesting a new URL instead of following hypertext links. It is possible to continue an interrupted conversation if a page corresponding to the interrupted conversation is still available to the client, e.g., in the browser cache or in disk memory. If so, the conversation may be continued by reloading the page and continuing to follow hyperlinks. A client may communicate with multiple servers during the same conversation.

More formally, a series of HTML pages p**1**, p**2**, . . . , pn constitutes a conversation if:

1. p**1**, p**2**, . . . , pn were all viewed by a client, and
2. for all i, such that 1<i<=n, page pi was obtained by following a hypertext link on page pi−1.

In an uninterrupted conversation, the client simply follows n−1 hypertext links to get from page p**1** to pn without ever "backtracking". In an interrupted conversation, the client backtracks at least once. By backtracking, we mean that the client:

1. Initially visits a page pi where 1<=i<n,
2. Views other pages either by following hyperlinks or explicitly accessing URL's, and
3. Returns to page pi by reloading pi from memory (presuming that pi is still available).

All requests for URL's are stateless. Even if a client requests a page multiple times, the server doesn't maintain any history or knowledge of previous connections. When a client requests an HTML file, there is no way for the client to communicate additional information with the request. Thus, a need exists in the Web environment to preserve state information throughout a conversation while a client is browsing HTML files. The present invention addresses such a need.

For example, consider a server which is handling business transactions. In order to function properly, the server needs state information such as the client's user ID and the transaction number corresponding to the current transaction number. Thus, there is a need to preserve this information while the client is browsing HTML files by following hyperlinks in a conversation. The present invention addresses such a need.

Current Methods for Handling State on the Web

One current method for handling state on the Web involves the use of CGI programs. A client can invoke a CCI program by passing arguments to it. For example, the command, http://tranman.watson.ibm.com/cgi-bin/get-args?var**1**=7 & var**2**=10 invokes a CGI program passing the variables var**1**=7 and var**2**=10. It is cumbersome to expect the client to follow the exact syntax for passing variables to CGI programs. A more user-friendly method is to allow the user to input arguments via an HTML "form". An example of an HTML form as displayed by a Web browser is shown in FIG. **2**. The user fills in the appropriate fields and sends the information to the server by clicking on the send button. The values typed in by the user are passed along as arguments to a CGI script. "Forms" provide a convenient interface for passing arguments to CGI programs. The client does not need to know the details of the CGI program being invoked or the format of the arguments expected by the program.

Forms allow the client to pass state variables to the server. Servers can also use forms to pass variables to the client. Forms may include hidden variables which are not displayed to clients and which are passed back to a server when the client submits the form. Web servers typically preserve state by passing state variables as hidden variables within forms. When the client submits the form, the server receiving the form can obtain the state variables from the hidden fields.

For example, suppose that a business transaction server is communicating with a client. The transaction server needs to obtain a client user ID and a session ID for the remainder of the conversation with the client. The server can obtain the client's user ID from a form submitted by the client. The form invokes a CGI program which then generates a session ID. Each subsequent response from the server is a form. The form is generated dynamically and contains the user and session ID's embedded as hidden variables. Clients respond by completing and submitting the forms generated by the server.

FIG. **3** depicts an example of a current method for preserving state using HTML forms. As depicted, the server **410** embeds state variables in hidden arguments to HTML forms **420** which are generated dynamically. The state variables **425** are passed back and forth between the client **450** and the server **410**. Using forms, the client **450** and the server **410** pass the state information **425** back and forth. The server **410** passes the state information to the client by creating HTML forms **420** on the fly and embedding the state variables **425** in hidden fields. The client **450** passes the state information **425** back to the server by completing and submitting the forms **420'** generated by the server **410**.

Limitations of the Current Technology for Handling State

The problem with the approach just outlined is that it seriously limits the types of interactions between a client and a server during a conversation. The server **410** must always respond to the client **450** with a dynamically generated HTML form **420** containing hidden variables **425**. There is no way to preserve state while the client browses HTML files. For example, suppose that the client wishes to browse a catalog in the middle of the session. The catalog consists of HTML files. There is no way to allow the client to browse (different HTML files in) the catalog without losing the state information using current technology. If the server allows the client to continue a conversation by viewing the catalog, the state information will be lost as soon as the client accesses an HTML file from the catalog.

Thus, there is a need for a system and method that allows the client to browse the catalog, i.e., access different HTML files while preserving the state information. The present invention addresses such a need, regardless of whether the HTML files constituting the catalog reside on different servers.

The limitations of the current technology for preserving state have been noted by others, see, e.g., "Persistent Client State HTTP Cookies", Netscape Communications Corporation, 1996, http://home.netscape.com/newsref/std/cookie_spec.html; see also, "Proposed HTTP State-Info Mechanism", D. M Kristol, AT&T Bell Laboratories, Sep. 22, 1995, http://www.research.att.com/~dmk/session01.txt.; and M. Cutler and D. Hall, "August 1995 Web Watch", http://www.netgen.com/corpinfo/press/webwtchv6n8.html. Unlike the solution suggested by Kristol which would modify the HTTP protocol to preserve state, the present invention preserves state without requiring changes to the underlying protocol.

Another solution, by Netscape Communications has been to add a feature called Cookies to their browsers; see

"Persistent Client State HTTP Cookies", Netscape Communications Corporation, 1996, http://home.netscape.com/newsref/std/cookie_spec.html. Here, a server can satisfy an HTTP request by appending a state object known as a cookie to its response. The cookie contains a description of the range of URL's for which the state is valid. The cookie is stored by the Netscape browser running on the client. Any future HTTP requests made by the client to one of the URL's specified in the cookie will include a transmittal of the state object stored in the cookie from the client back to the server.

There are a number of drawbacks to this approach. The server application which wishes to preserve state must provide a list of all URL's which might make use of the state. This is cumbersome and may sometimes be impossible. Cookies also lack a method for correlating state information with specific conversations. For example, suppose a browser accesses the same URL in two separate conversations. During the first conversation, state information exists at the time the URL is accessed and is passed to the server via a cookie. During the second conversation, no state information exists at the time the URL is accessed. However, the old cookie still exists and the old state is still passed back to the server. This would confuse the server into believing that the old state information still applies to the new conversation. Another problem is that cookies are not a standard feature and will only work with servers and browsers which support Netscape's protocol.

Thus, there is a need for a method and system for preserving state in a stateless protocol which is not limited to a list of URL's which need to make use of the state information and where state information is correlated with specific conversations to avoid the problem of passing outdated state information to a server. Moreover, there is a need for a system of preserving state in a protocol as HTTP that works with any browser supporting the HTTP protocol and doesn't require specialized nonstandard features on the client or server.

SUMMARY OF THE INVENTION

The present invention, in accordance with the aforementioned needs, is directed to a method and system for preserving state in computers communicating over networks using stateless protocols. Although, the preferred embodiment is for computers communicating over the World Wide Web (WWW or Web) using the Hypertext Transfer Protocol (HTTP), the present invention applies to other forms of networked communication as well.

It is assumed that the services performed by the server on behalf of a client are programs which the client invokes. A service can accept a variable number of arguments. A conversation is a sequence of communications between the client and one or more servers for services wherein each response from the server includes one or more continuations which enable another request for services and wherein the client must invoke one of the continuations to continue the conversation.

Accordingly, a computerized method, system, and computer program product having features of the present invention which preserves state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client, includes: the client initiating the conversation with the server using the stateless protocol; detecting when the request for a service requires preservation of the state information; performing the service and identifying all continuations in an output from the service, when state is to be preserved; recursively embedding the state information in all identified continuations; and communicating the output to the client; wherein the state information is preserved and provided to all services for the duration of the conversation.

According to another aspect of the present invention, the embedding of state information is performed by the server and communicated by the server to the client. Another aspect of the present invention includes storing at least part of the state information in a memory coupled to the server and embedding an index representing the part of the state information in all identified continuations.

Still another aspect of the present invention includes dynamically downloading computer program code to the client to embed the state information in the output from the service which is also communicated to the client. Yet another aspect of the present invention includes storing at least part of the state information in a memory coupled to the client and embedding an index representing the stored state information.

In a preferred embodiment, our method allows state to be preserved while traversing hypertext links using a Web browser on the World Wide Web. Hypertext links constitute continuations. A client browser follows a conversation by following hypertext links to fetch new pages. The present invention has features which preserves state variables across any conversation. According to one aspect of the present invention, state variables to be preserved throughout a conversation, are passed to every CGI program which could be invoked throughout the conversation.

When the client and the server are networked via the World Wide Web, the stateless protocol is the hypertext transfer protocol (HTTP), and the continuations are hyperlinks to one of hypertext markup language (HTML) files and common gateway interface (CGI) programs, the present invention has features which enable the filtering and/or addition of hyperlinks and data output from the services according to a predetermined criteria. Yet another aspect of the present invention for embedding state information includes modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments. Still another aspect of the present invention for embedding state information includes modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein the embedding step is performed by the CGI program. Another aspect of the present invention for embedding state information includes modifying an identified continuation which is an invocation to a CGI program to invoke a CGI converter program with the identified continuation, an argument counter which indicates a number of arguments associated with the CGI program, and the state information passed as arguments, wherein the embedding step is performed by the converter program.

III. BRIEF DESCRIPTION OF THE DRAWINGS

These and other features and advantages of the present invention will become apparent from the following detailed description taken in conjunction with the accompanying drawings, wherein:

FIG. **1** is an example of an HTML page as displayed by a standard browser;

FIG. **2** depicts an example of an HTML "form" as viewed by a Web browser;

FIG. **3** shows a block diagram of a client and server using a "form" to preserve state variables;

FIG. **4** is a block diagram of a general method for preserving state according to the present invention on a network using a stateless protocol;

FIG. **5** is a generalized diagram of a data packet for transmission via a stateless protocol;

FIG. **6** is an example of the data packet of FIG. **5** modified in accordance with the present invention to preserve state information;

FIG. **7***a* depicts an embodiment of a system having features of the present invention for transacting business over the World Wide Web while preserving state;

FIG. **7***b* depicts an embodiment of a method for preserving state on the system of FIG. **7***a;*

FIG. **8** depicts a more detailed example of a method according to the present invention for preserving state in the system shown in FIG. **7***a*, and method step **745** of FIG. **7***b;*

FIG. **9***a* depicts a structure of a hypertext link to an HTML file;

FIG. **9***a'* depicts the structure of FIG. **9***a* modified to preserve state in accordance with the present invention;

FIG. **9***b* depicts a structure of a hypertext link to a type 1 CGI program;

FIG. **9***b'* depicts the structure of FIG. **9***b* with embedded state arguments in accordance with the present invention;

FIG. **9***c* depicts a structure of a hypertext link to a type 2 CGI program; and

FIG. **9***c'* depicts the structure of FIG. **9***c* modified to preserve state in accordance with the present invention.

DETAILED DESCRIPTION OF A METHOD FOR PRESERVING STATE IN A CONVERSATION USING A STATELESS PROTOCOL

The present invention is an enabling technology for computers communicating over networks via stateless protocols. Although the preferred embodiment is for computers communicating over the World Wide Web (WWW or Web) using the Hypertext Transfer Protocol (HTTP), the present invention also applies to other forms of networked communication as well.

FIG. **4** depicts a general method in accordance with the present invention for preserving state using a stateless protocol, i.e., it is not restricted to clients and servers communicating over the World Wide Web. The preferred embodiment, which is described later, is specifically applied to the World Wide Web. Here, it is assumed, for simplicity, that the services performed by the server on behalf of a client are programs which the client invokes and that a service can accept a variable number of arguments.

As depicted, in step **500**, a client requests a service from a server. The path represented by steps **505**, **510**, and **515** would be taken when the services provided do not require state preservation. In step **510**, at some point the server processes a request for which the server determines that state variables need to be made available to all services which could be invoked in the current conversation. The server then passes its output and all of the state variables denoted by <state-variable-list> to a program denoted by convert**1** and the process continues to step **520**. In step **520**, the convert**1** program modifies the continuations produced by the service but passes back all other data to the client unmodified.

For example, as depicted in FIG. **5**, under normal circumstances, a continuation representing a call to a program service**1** would be of the form:

service**1**<service-arg-list> (1)

where service**1** **605** is a service and <service-arg-list> is the list of service arguments **610** passed to the service if the client chooses the continuation. As depicted in FIG. **6**, the convert**1** program of the present invention preserves state by modifying each continuation (1) to be of the form:

convert**2** service-string <state-variable-list>, (2)

where convert**2** **650** is a call to a special service (which will be described later), and service-string **680** is a string containing service**1** **605** and <service-arg-list> **610** and some delimiting information **685** to distinguish the service arguments **610** from <state-variable-list> **670** in the call to convert**2** shown in FIG. **6**. The <state-variable-list> **670** represents the state information to be preserved and made available to all services for the duration of the conversation.

Referring again to FIG. **4**, in step **525**, the client receives the output and modified continuations sent from the server. Each modified continuation for the conversation is now a call to the convert**2** program, as in (2). In step **540**, the client examines the output. If a continuation is selected, the process returns to step **500** where the (modified) service request is sent to the server. In step **505**, the server processes the modified service request and invokes the convert**2** program and processing continues at step **530** (due to the explicit embedded call of prior step **520**). In step **530**, (with reference to FIGS. **5** and **6**) the convert**2** program parses the service **605** and the arguments to be passed to the server (<service-arg-list>) **610** from service-string **680**. The convert**2** program **650** invokes the requested service **605** (here, service**1**) by passing it all variables on <service-arg-list> **610** as well as <state-variable-list>. That way, service**1** has access to all state variables, as needed. In step **535**, the convert**2** program receives the service (service**1**) output and passes the output and the <state-variable-list> to convert**1**. In step **520**, convert**1** modifies each of the continuations as discussed above. The output is again communicated to the client in step **525** and the process repeats with the state information **670** preserved for the duration of the conversation.

The Preferred Embodiment

FIG. **7***a* depicts an embodiment of a system having features of the present invention for transacting business over the World Wide Web. A Web server **410'** allows businesses to sell goods over the Internet. Customers access the server **410'** via a client **450** running a standard browser **460**. In order to communicate securely, the 'browser' **460** should be able to communicate using SSL. See, e.g., A. O. Freier, P. Karlton, P. C. Kocher. "The SSL Protocol Version 3.0", Internet Draft, March 1996, http://home.netscape.com/eng/ssl3/ssl-toc.html, which is hereby incorporated by reference in its entirety. However, some services can be used by browsers which don't support SSL. Users may browse catalogs which may be stored on a stable storage medium such as direct access storage device (DASD) **470**. As with conventional catalogs, users browse product descriptions and can pick and choose which items to add to their purchase lists. When the user has determined that the purchase list is complete, he commits to the purchase and is subsequently billed.

As depicted, the server **410'** may include a traditional database management system (DBMS) **412** to manage information about the customer, inventory, and products stored in the database **475**. An exemplary DBMS is that sold by IBM under the trademark 'DB2'. In addition, the server **410'** allows users to browse product catalogs in the course of a

conversation. The server **410'** assumes very little about the format of product catalogs. Catalogs may consist of HTML files **425** as well as conventional CGI programs. The files and/or programs may be associated with local or remote servers. State information, e.g., a User-ID and session-ID must be maintained between the server **410'** and a client **450** during conversations. The present invention provides an improved method and system to transparently maintain this state information during a conversation.

Any client **450** may access a 'home page' associated with the server **410'** as well as view product catalogs. In order to purchase products, update customer information, or access certain types of information, it is necessary for the user to provide authentication by entering a User-Id and password. According to the present invention, authentication is only required once per conversation. As soon as a user has been authenticated, the user-id is embedded (preserved) into the conversation by the converter **416** of the present invention.

FIG. **7***b* depicts an example of a method according to the present invention for a client **450** to interact with the server **410'** using HTTP while preserving state. As depicted, in step **700**, the client accesses a home page residing on server **410'**. In step **710**, the client begins browsing a product catalog and in step **720**, continues browsing the catalog offerings, e.g., by selecting hyperlinks from the on-line product catalog. Since no authentication is needed to merely browse the catalog, communication is stateless and the number of people which may browse the catalog is maximized. In step **730**, an item is found which is to be added to a purchase list. In step **740**, the client must then enter a user ID and password to continue. If the client is new to the system, the client picks a user-ID, password, and provides some additional optional information to the server (address, phone number, etc.). In step **745**, the converter **416** embeds the user-ID and session-ID into the conversation in accordance with the present invention. In step **750**, the user can view additional products, add additional items to the purchase list, commit to purchases, or view and update database information. The state variables are advantageously preserved and re-authentication is not required. The state information, i.e., the user-ID and session-ID will be preserved and made available to every CGI program which is invoked during the remainder of the conversation.

Recall that using current "forms" technology, the user would have to re-enter the user-ID and password each time an action requiring authentication such as adding a new item to the purchase list was attempted. The session-ID would present even greater difficulties in that the server would have to tell the client to remember the session-ID and enter it whenever authentication is needed.

Recall also that using "cookies" limits the range of URL's for which state is preserved. Using cookies further lacks the ability to correlate state information with specific conversations which may cause outdated state information to be provided a server. Lastly, cookies require the use of a specific browser and may require specialized and/or non-standard features on the client or server.

FIG. **8** depicts a more detailed example of a method according to the present invention for preserving state in the system shown in FIG. **7***a*, and method step **745** of FIG. **7***b*. Assume that server **410'** is a conventional Web server including typical Internet connections and access such as TCP/IP couplings and further has HTML and Common Gateway Interface (CGI) capabilities **413**.

As depicted, in step **740'**, assume a client **450** running Web browser **460** selected a hyperlink to request a service via (stateless protocol) HTTP to a Web server **410'**. In step **810**, the server **410'** interprets the URL, for example, as being a call to a CGI program 'p**1**' **415** which determines that state variables, e.g., 'x**1**, x**2**, . . . , xn', should be embedded in the conversation so that all CGI programs which could be invoked from the conversation are given access. P**1** generates an HTML page 'h' with hypertext links for the client **450** to continue the conversation. Instead of returning the output, page 'h' to the client unmodified, 'p**1**' has been adapted to invoke the converter program **416** of the present invention by passing to a convert**1** module of the converter **416** the arguments 'h, x**1**, x**2**, . . . , xn'. The call to convert**1** could be of the form:

convert**1** 'h, x**1**, x**2**, . . . , xn'

In steps **811** and **812**, the convert**1** module of the converter program modifies all the hypertext links to HTML in h, to preserve the state variables. All relative hypertext links are converted to absolute hypertext links (also called hyperlinks). See the aforementioned and incorporated by reference U.S. Pat. No. 5,752,022, issued May 12, 1998 to Chiu et al., for an example of a relative to absolute address conversion scheme. As noted, these applications have a common assignee, International Business Machines Corporation, Armonk, N.Y. Those skilled in the art will appreciate that the modification of links to HTML files (step **811**) and links to CGI programs (step **812**) could be done in a one-pass or a two-pass process within the scope and spirit of the present invention.

As depicted, in step **811** the convert**1** module of the converter program **416** takes HTML page h and modifies all the hypertext links to HTML files to preserve the state variables. Hypertext links to HTML files may be modified to be a call to a CGI program convert**2** with arguments consisting of h, x**1**, . . . , xn. With reference to FIG. **9***a*, consider, for example that h contains the following reference to an HTML file **915** "mail.html", and suppose that the state variables **930** were x=32 and y=45:

<A HREF="http://www.watson.ibm.com/mail.html">

would be modified by the convert**1** logic to the form depicted in FIG. **9***a'*:

<A HREF="http://www.watson.ibm.com/cgi-bin/convert**2**?url=//
www.watson.ibm.com/mail.html&x=32& y=45">

In step **812**, the convert**1** module of the converter program modifies all the hypertext links to CGI programs. Note that hypertext links which are calls to CGI programs may have the state variables preserved two different ways:

(a) As depicted in FIG. **9***b'*, pass the state variables **930'** to the CGI program **940** but don't embed the state variables within any hypertext links generated by the CGI program, i.e., don't embed a call to convert**2**. Using this approach, the CGI program is responsible for propagating state within hypertext links it generates; or

(b) Preferably, as depicted in FIG. **9***c'*, pass the state variables **930"** to the CGI program **950** and embed the state variables (by an embedded call to the converter **920'**) within hypertext links generated by the CGI program.

In order to take advantage of both approaches (a) and (b), the converter may determine how to distinguish CGI programs based on any one of a variety of techniques within the scope of the present invention. As depicted in FIG. **9***b*, for example, a naming convention could be used whereby any

CGI program whose name begins with the substring "type" may be considered a type I CGI program and is processed using the first method (a). Any substring whose name does not begin with the substring "type" may be considered a type II CCI program and is processed using the second method (b).

For example, consider (with reference to FIG. **9***b*) the following example of a type 1 CGI call:

<A HREF="http://www.watson.ibm.com/cgi-bin/type**1**?arg**1**=55">

Suppose that the state variables are x=32 and y=45. The converter **416** would append the state variables **930'** to the hypertext link to the following form (as depicted in FIG. **9***b'*):

<A HREF="http://www.watson.ibm.com/cgi-bin/type**1**?&arg**1**=55&x=32&y=45">

Now consider, with reference to FIG. **9***c*, an example of a hypertext link to a type 2 CGI program:

<A HREF="http://www.watson.ibm.com/cgi-bin/prog?arg**1**=55">

Suppose again that the state variables **930"** are x=32 and y=45. The converter would modify this hypertext link to the following form (as depicted in FIG. **9***c'*):

<A HREF="http://www.watson.ibm.com/cgi-bin/convert**2**?url=//www.watson.ibm.com/cgi-bin/prog&numargs=1&argl=55&x=32&y=45">

where the "numargs=1" argument **970** indicates to convert**2** that the CGI program **950** initially only had 1 argument passed to it and the remaining arguments are state variables **930"** passed by the converter. The modified output is then returned to the requesting client. In step **813**, the client **450** receives HTML file h from the server **410'**. Every hypertext link (with the exception of hypertext links resulting from type 1 CGI programs) returned to the client is now a call to the convert**2** routine of the converter. In step **814** the client **450** running browser **460** selects one of the hypertext links. In step **815**, the server determines if the selected hypertext link is a call to convert**2**. If yes, the process continues at step **817**. In step **817**, there are two possibilities:

(1) The URL passed to convert**2** references an HTML file. Here, the process continues at step **819**. Suppose, for example, the client had selected the following link:

<A HREF="http://www.watson.ibm.com/cgi-bin/convert**2**?url=//www.watson.ibm.com/mail.html&x=32&y=45">

In step **819**, convert**2** fetches the HTML page contained in the file "mail.html". It then passes the HTML page and the state arguments x=32 y=45 to the convert**1** module of the converter and the process returns to step **811**, as described previously, or,

(2) The hypertext link is a call to a CGI program. In this case, the process continues to step **818**. Suppose, for example, the client had selected the following link:

<A HREF="http://www.watson.ibm.com/cgi-bin/convert**2**?url=//www.watson.ibm.com/cgi-bin/prog&numargs=1&arg**1**=55&x=32&y=45">

Here, the second argument to convert**2**, numargs=1, indicates that the initial hypertext link only passed one argument to "prog", i.e., "arg**1**=55". The other two arguments, "x=32" and "y=45", are state variables which were embedded by the converter **416**. Convert**2** passes all three arguments to prog, including the state variables. The process then returns to step **811**, as described previously.

This method of the present invention advantageously preserves state information by embedding the state in all hyperlinks passed back and forth between the client **450** and server **410**. Those skilled in the art will appreciate that, the level of detail which is communicated between the client and server may be reduced by storing most of the state information in a file system or a database **425** coupled to the server **410'**. In this case, it is only necessary to pass an index (or pointer) to the state variables back and forth between the client and server.

The present invention is designed to work for a standard browser **460** which doesn't necessarily support downloading of programs from the server which can then execute on the client. For a browser which supports downloadable server programs such as those written using Java ("applets"), or any other such language, additional features are possible. The Java programming environment is well known in the art. See for example, P. Tyma, G. Torok, and T. Downing. "Java Primer Plus", Waite Group Press, 1996, which is hereby incorporated by reference in its entirety. See also Patrick Naughton, "The Java Handbook" Osborne Mcgraw-Hill, 1996, which is hereby incorporated by reference in its entirety. For example, the server **410'** could contain a downloadable program which causes the state to be stored at the client. Using this approach, all or part of the state could be stored on the client. An index referencing the location of the state information in memory, as noted above, may be passed back and forth between the server and client to allow the state to be retrieved from the client.

Another application of downloadable server code to the present invention would be to allow the 'converter' **416** to run on the client. Here, clients would download all or part of the 'converter' logic **416** from the server **410'** to the client for execution. This would allow the full functionality of the present invention with all (or part of) the processing taking place locally on the client **450**. The client no longer has to go through a remote server to filter HTML pages during a conversation; all of the filtering takes place locally. An advantage here is that the load on the server is reduced. In addition, the client will be able to continue conversations even if the server from which the client obtains the applet goes down or becomes unavailable due to a network failure.

Other Embodiments

Preserving State on Multiple Communicating Servers

Those skilled in the art will appreciate that within the scope of the present invention multiple converters may be used for state propagation on multiple servers. For example, an airline reservation system over the Web might have a converter (converter A) for maintaining state. One of the hypertext links might be to a hotel booking system on a remote server with its own converter (converter H). A client might begin using the airline reservation system. At some point, state information is attached to the conversation. The client then follows a hypertext link to the hotel booking system. Converter A continues to maintain state information while the client is using the hotel booking system. All state variables are propagated to the hotel booking system's CGI programs. These remote server CGI programs might simply ignore these state variables. On the other hand, if the hotel booking system understands the state variables from the airline reservation systems, these variables could be used by the hotel booking system (converter H).

At some point, the hotel reservation system server may invoke its converter (converter H) to embed additional state variables. When this happens, the call to converter H may be nested within the call to converter A. This will not present problems. CGI programs will now be passed arguments

from both converter A and converter H. If converter A has the ability to recognize a CGI function representing a call to converter H, additional things are possible:

(1) Converter A could treat converter H as a type I CGI program as discussed previously. In this case, converter A can stop monitoring future hypertext links in the conversation.

(2) Converter A could treat converter H as a type II CCI program and continue to modify hypertext links. In addition, converter A could add special links to future HTML pages which would allow the user to escape from the control of either converter.

Other Examples of Dynamic Page Modification

The present invention also has features which provide a system and method for filtering all HTML text viewed by a client while the client is browsing HTML files in a conversation. For example, suppose that a client has contacted a server and started a conversation. The server wishes to filter all HTML text and leave out phrases and hypertext links which have been determined to be objectionable. The present invention provides a method for filtering and/or modifying HTML text while a client accesses files and programs which may be remote to the server doing the filtering.

The present invention can be applied to a wide variety of applications where HTML pages need to be modified during a conversation. For example, suppose that a server application wishes to filter all HTML pages which are passed to a client in a conversation. The converter could modify and/or remove undesirable parts of HTML pages before sending them to the client. The converter would merely have to be modified to search text for different substrings. Note that the converter can censor pages and output from CCI programs which reside on remote servers. If the client can download programs from the server written in a language such as Java, the converter doing the censoring can execute on the client.

As another example, suppose that a client **450** is in a conversation where the names of major corporations appear frequently in the text. A server **410'** running a converter **416** has access to a database **475** of home page URL's for major corporations. The server wishes to add hypertext links each time the name of a company in the database appears in an HTML page. For example, each time the name IBM or International Business Machines appears in an HTML page, the server would like to convert the reference to a hypertext link to IBM's home page. By doing this, the client would be able to obtain useful information about companies appearing in the conversation by pointing and clicking. This can be accomplished by modifying the converter **416** to search HTML pages for all company names which appear in the database. Whenever such a name is found, a hypertext link to the company's home page would be inserted into the HTML text returned to the client. The converter **416** can continue to monitor the conversation in the event that hypertext links are followed to remote servers. As noted above, if the client can download programs from the server written in a language such as Java, the converter can execute on the client.

Now that the present invention has been described by way of a preferred embodiment, with alternatives, various improvements will occur to those of skill in the art. Thus, it should be understood that the preferred embodiment has been provided as an example and not as a limitation. The scope of the invention is properly defined by the appended claims.

What is claimed is:

1. A computerized method for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client, said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation, the method comprising the steps of:

the client initiating the conversation with the server using the stateless protocol;

detecting when the request for a service requires preservation of the state information;

performing said service and identifying all continuations in an output from said service, in response to said step of detecting;

recursively embedding the state information in all identified continuations; and

communicating the output to the client, in response to said step of embedding; wherein the state information is preserved and provided to all services for the duration of the conversation.

2. The method of claim **1**, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding.

3. The method of claim **2**, further comprising the step of storing at least part of the state information in a memory coupled to the server and wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations.

4. The method of claim **1**, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client.

5. The method of claim **4**, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information.

6. The method of claim **1**, further comprising the steps of:

the client selecting a second continuation from said all identified continuations with embedded state information; and

restoring the state information from said second continuation and invoking an associated second service with restored state information;

recursively identifying and embedding the state information in all continuations associated with an output from said second service.

7. The method of claim **1**, further comprising the step of correlating the state information to a specific conversation.

8. The method of claim **1**, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.

9. The method of claim **8**, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.

10. The method of claim **8**, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.

11. The method of claim **8**, wherein said step of embedding further comprises the step of:

modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program

with the identified continuation and the state information passed as arguments.

12. The method of claim **8**, wherein said step of embedding further comprises the step of:

modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program.

13. The method of claim **8**, wherein said step of embedding further comprises the step of:

modifying an identified continuation which is an invocation to a CGI program to invoke a CGI converter program with the identified continuation, an argument counter which indicates a number of arguments associated with the CGI program, and the state information passed as arguments, wherein said step of embedding is performed by the converter program.

14. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to provide a method for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client, said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation, the method comprising the steps of:

the client initiating the conversation with the server using the stateless protocol;

detecting when the request for a service requires preservation of the state information;

performing said service and identifying all continuations in an output from said service, in response to said step of detecting;

recursively embedding the state information in all identified continuation; and

communicating the output to the client, in response to said step of embedding; wherein the state information is preserved and provided to all services for the duration of the conversation.

15. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **14**, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding.

16. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **15**, further comprising the step of storing at least part of the state information in a memory coupled to the server and wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations.

17. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **14**, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client.

18. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **14**, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information.

19. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **14**, further comprising the steps of:

the client selecting a second continuation from said all identified continuations with embedded state information; and

restoring the state information from said second continuation and invoking an associated second service with restored state information;

recursively identifying and embedding the state information in all continuations associated with an output from said second service.

20. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **14**, further comprising the step of correlating the state information to a specific conversation.

21. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **14**, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.

22. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **21**, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.

23. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **21**, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.

24. The program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **21**, wherein said step of embedding further comprises the step of:

modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments.

25. The program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **21**, wherein said step of embedding further comprises the step of:

modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program.

26. The program storage device readable by a computer, tangibly embodying a program of instructions executable by

the computer to perform method steps as claimed in claim **21**, wherein said step of embedding further comprises the step of:

modifying an identified continuation which is an invocation to a CGI program to invoke a CGI converter program with the identified continuation, an argument counter which indicates a number of arguments associated with the CGI program, and the state information passed as arguments, wherein said step of embedding is performed by the converter program.

27. A computer system for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client, said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services, wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation, the system comprising:

the client being adapted for initiating a conversation with the server using the stateless protocol;

state detection logic for detecting when the request for a service requires preservation of the state information;

search logic for identifying all continuations in an output from said service, in response to said step of detecting;

converter logic for recursively embedding the state information in all identified continuations; and

communication logic for communicating the output to the client; wherein the state information is preserved and provided to all services for the duration of the conversation.

28. The computer system of claim **27**, wherein said converter logic is executed by the server and said communication logic communicates the output with embedded state information from the server to the client.

29. The computer system of claim **28**, further comprising:

a memory, coupled to the server, for storing at least part of the state information; wherein said converter logic is adapted for embedding an index representing said part of the state information in said all identified continuations.

30. The computer system of claim **27**, wherein said communication logic communicates the output without embedded state information from the server to the client; and wherein the server is adapted for dynamically downloading said converter logic to the client for execution.

31. The computer system of claim **30**, further comprising:

a memory, coupled to the client, for storing at least part of the state information; wherein said converter logic is further adapted for embedding an index representing said part of the state information.

32. The computer system of claim **27**, wherein the client selects a second continuation from said all identified continuations with embedded state information, further comprising:

the converter logic being further adapted for restoring the state information from said second continuation, invoking an associated second service with restored state information, and recursively identifying and embedding the state information in all continuations associated with an output from said second service.

33. The computer system of claim **27**, wherein the state information is correlated to a specific conversation.

34. The computer system of claim **27**, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.

35. The computer system of claim **34**, further comprising filter logic for filtering one of said hyperlinks and data output from said services according to a predetermined criteria.

36. The computer system of claim **34**, further comprising integration logic for adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.

37. The computer system of claim **34**, wherein said step of embedding further comprises the step of:

modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments.

38. The computer system of claim **34**, wherein said step of embedding further comprises the step of:

modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program.

39. The computer system of claim **34**, wherein said step of embedding further comprises the step of:

modifying an identified continuation which is an invocation to a CGI program to invoke a CGI converter program with the identified continuation, an argument counter which indicates a number of arguments associated with the CGI program, and the state information passed as arguments, wherein said step of embedding is performed by the converter program.

40. A computer system for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client, said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation, the system comprising:

the client being adapted for initiating the conversation with the server using the stateless protocol;

state detection means for detecting when the request for a service requires preservation of the state information;

search means for identifying all continuations in an output from said service, in response to said step of detecting;

converter means for recursively embedding the state information in all identified continuations; and

communication means for communicating the output to the client; wherein the state information is preserved and provided to all services for the duration of the conversation.

41. The computer system of claim **40**, wherein said converter means is executed by the server and said communication means communicates the output with embedded state information from the server to the client.

42. The computer system of claim **41**, further comprising:

a memory, coupled to the server, for storing at least part of the state information; wherein said converter means is adapted for embedding an index representing said part of the state information in said all identified continuations.

43. The computer system of claim **40**, wherein said communication means communicates the output without

embedded state information from the server to the client; and wherein the server is adapted for dynamically downloading said converter means to the client for execution.

44. The computer system of claim **43**, further comprising:

a memory, coupled to the client, for storing at least part of the state information; wherein said converter means is further adapted for embedding an index representing said part of the state information.

45. The computer system of claim **41**, wherein the client selects a second continuation from said all identified continuations with embedded state information, further comprising:

the converter means being further adapted for restoring the state information from said second continuation, invoking an associated second service with restored state information, and recursively identifying and embedding the state information in all continuations associated with an output from said second service.

46. The computer system of claim **45**, further comprising integration means for adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.

47. The computer system of claim **40**, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.

48. The system of claim **47**, wherein said converter means further comprises:

means for modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments.

49. The system of claim **47**, wherein said converter means further comprises:

means for modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said converter means comprises the CGI program.

50. The system of claim **47**, wherein converter means further comprises:

means for modifying an identified continuation which is an invocation to a CGI program to invoke a CGI converter program with the identified continuation, an argument counter which indicates a number of arguments associated with the CGI program, and the state information passed as arguments, wherein said converter means comprises the converter program.

51. A computerized method for preserving state information in a conversation via a stateless protocol between a client adapted to request services from one or more servers, the method comprising the steps of:

receiving a service request including state information, via the stateless protocol;

identifying all continuations in an output from said service and recursively embedding the state information in all identified continuations, in response to said request; and

communicating a response including the continuations and embedded state information, wherein the continuations enable another service request and one of the continuations must be invoked to continue the conversation.

52. The method of claim **51**, wherein said embedding is performed by a server and said step of communicating is in response to said embedding step.

53. The method of claim **52**, further comprising the step of storing at least part of the state information in a memory coupled to the server and wherein embedding step includes embedding an index representing said part of the state information in said all identified continuations.

54. The method of claim **51**, further comprising the step of dynamically downloading computer program code to the client to perform said embedding step, in response to said step of communicating the output to the client.

55. The method of claim **54**, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said embedding step includes embedding an index representing said part of the state information.

56. The method of claim **51**, further comprising the steps of:

receiving a second request associated with a second continuation from said all identified continuations with embedded state information; and

restoring the state information from said second continuation and invoking an associated second service with restored state information;

recursively identifying and embedding the state information in all continuations associated with an output from said second service.

57. The method of claim **51**, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.

58. The method of claim **57**, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.

59. The method of claim **57**, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.

60. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to provide a method for preserving state information in a conversation via a stateless protocol between a client adapted to request services from one or more servers, the method comprising the steps of:

receiving a service request including state information, via the stateless protocol;

identifying all continuations in an output from said service and recursively embedding the state information in all identified continuations, in response to said request; and

communicating a response including the continuations and embedded state information, wherein the continuations enable another service request and one of the continuations must be invoked to continue the conversation.

61. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **60**, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding.

62. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **60**, further comprising the step of storing at least part of the state information in a memory coupled to the server and wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations.

63. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **60**, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client.

64. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **60**, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information.

65. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **60**, further comprising the steps of:

receiving a second request associated with a second continuation from said all identified continuations with embedded state information; and

restoring the state information from said second continuation and invoking an associated second service with restored state information;

recursively identifying and embedding the state information in all continuations associated with an output from said second service.

66. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **60**, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.

67. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **66**, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.

68. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim **66**, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.

* * * * *

EXHIBIT D

US007631346B2

(12) **United States Patent**
Hinton et al.

(10) **Patent No.: US 7,631,346 B2**
(45) **Date of Patent: Dec. 8, 2009**

(54) **METHOD AND SYSTEM FOR A RUNTIME USER ACCOUNT CREATION OPERATION WITHIN A SINGLE-SIGN-ON PROCESS IN A FEDERATED COMPUTING ENVIRONMENT**

(75) Inventors: **Heather Maria Hinton**, Austin, TX (US); **Ivan Matthew Milman**, Austin, TX (US); **Venkat Raghavan**, Austin, TX (US); **Shane Bradley Weeden**, Gold Coast (AU)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 827 days.

(21) Appl. No.: **11/097,587**

(22) Filed: **Apr. 1, 2005**

(65) **Prior Publication Data**

US 2006/0236382 A1 Oct. 19, 2006

(51) **Int. Cl.**

G06F 7/04	(2006.01)
G06F 15/16	(2006.01)
G04L 9/32	(2006.01)
G06F 17/30	(2006.01)

(52) **U.S. Cl.** **726/8**; 380/279

(58) **Field of Classification Search** 726/6, 726/5, 4, 8; 709/223, 229, 219; 713/202, 713/186, 169; 380/279; 715/500
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,290,278	B2 *	10/2007	Cahill et al.	726/6
2003/0149781	A1	8/2003	Yared et al.	
2003/0154266	A1 *	8/2003	Bobick et al.	709/223
2004/0010607	A1 *	1/2004	Lee et al.	709/229
2004/0158746	A1 *	8/2004	Hu et al.	713/202
2004/0205176	A1 *	10/2004	Ting et al.	709/223
2005/0074126	A1 *	4/2005	Stanko	380/279
2005/0210270	A1 *	9/2005	Rohatgi et al.	713/186
2005/0240763	A9 *	10/2005	Bhat et al.	713/169
2005/0257130	A1 *	11/2005	Ito	715/500.1
2006/0048213	A1 *	3/2006	Cheng et al.	726/5

(Continued)

OTHER PUBLICATIONS

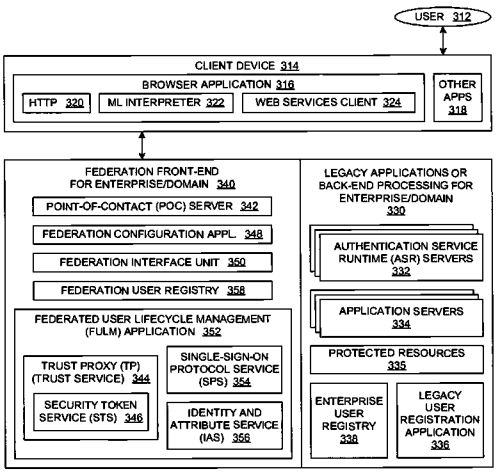
Gross, T.; Security analysis of the SAML single sign-on browser/artifact profile; Publication Date: Dec. 8-12, 2003; IBM Zurich Res. Lab; On pp. 298-307.*

Primary Examiner—Kambiz Zand
Assistant Examiner—Monjour Rahim
(74) *Attorney, Agent, or Firm*—Jeffrey S. LaBaw; David H. Judson

(57) **ABSTRACT**

A method, system, apparatus, and computer program product are presented to support computing systems of different enterprises that interact within a federated computing environment. Federated single-sign-on operations can be initiated at the computing systems of federation partners on behalf of a user even though the user has not established a user account at a federation partner prior to the initiation of the single-sign-on operation. For example, an identity provider can initiate a single-sign-on operation at a service provider while attempting to obtain access to a controlled resource on behalf of a user. When the service provider recognizes that it does not have a linked user account for the user that allows for a single-sign-on operation with the identity provider, the service provider creates a local user account. The service provider can also pull user attributes from the identity provider as necessary to perform the user account creation operation.

20 Claims, 14 Drawing Sheets



U.S. PATENT DOCUMENTS

2006/0059544 A1* 3/2006 Guthrie et al. 726/4
2006/0195893 A1* 8/2006 Caceres et al. 726/8
2007/0005730 A1* 1/2007 Torvinen et al. 709/219

* cited by examiner

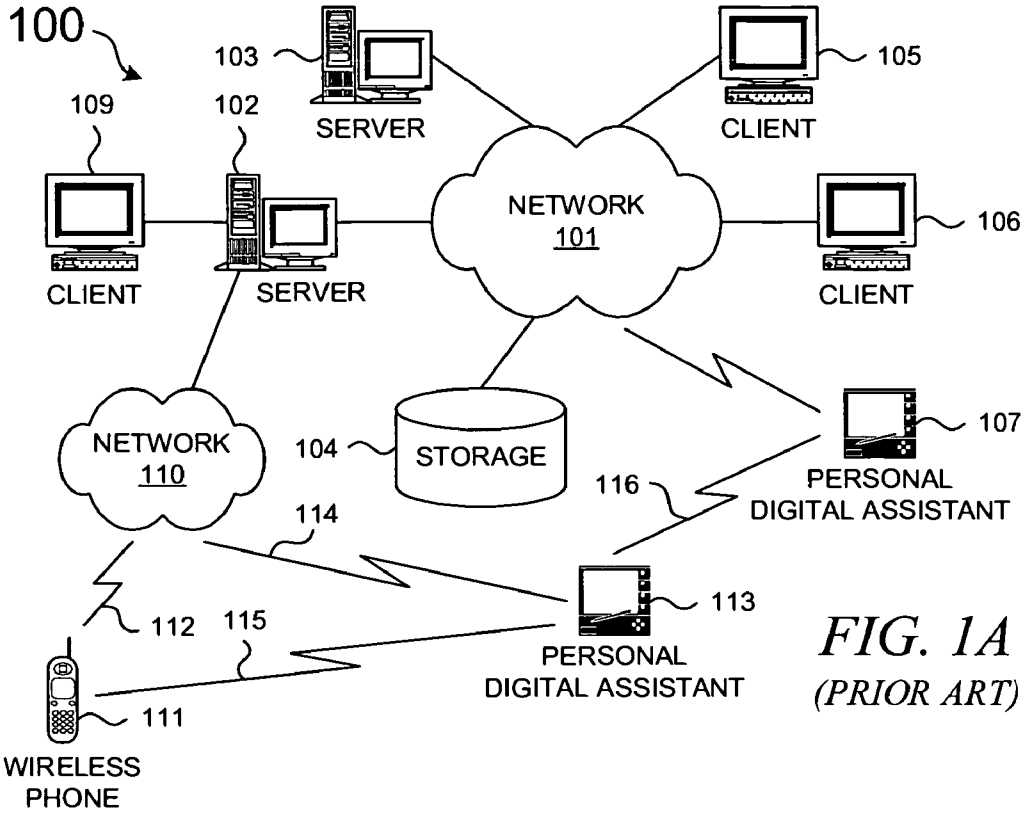


FIG. 1A
(PRIOR ART)

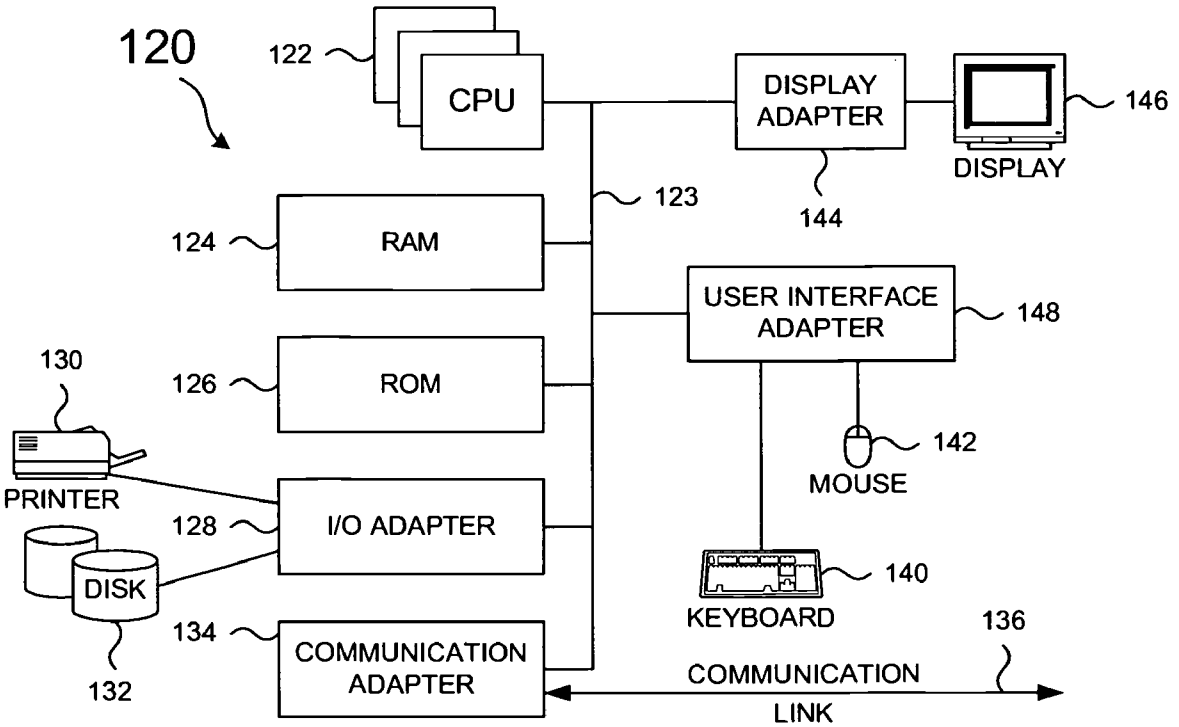


FIG. 1B
(PRIOR ART)

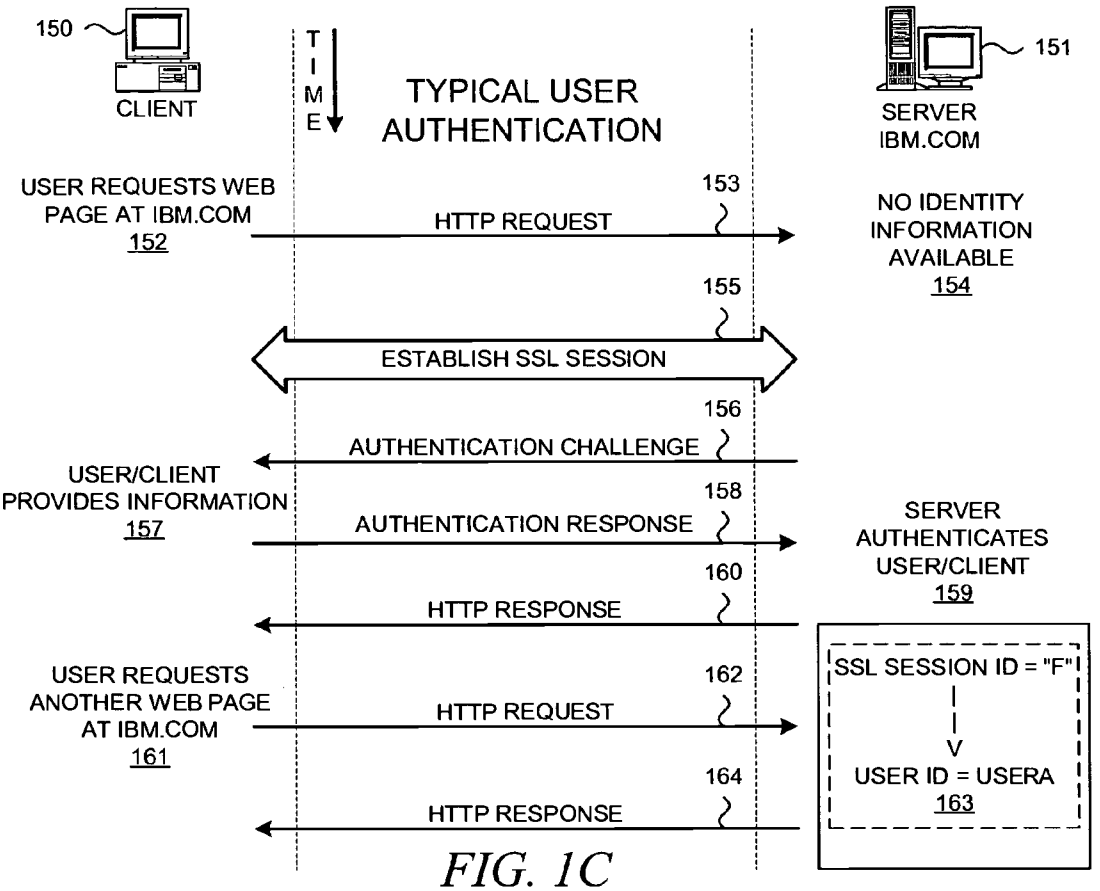


FIG. 1C
(PRIOR ART)

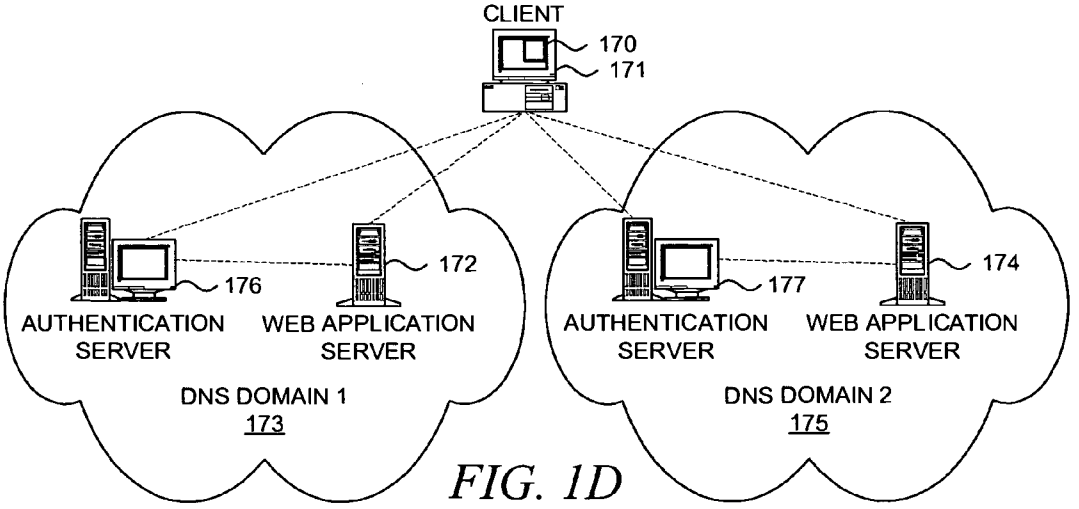


FIG. 1D
(PRIOR ART)

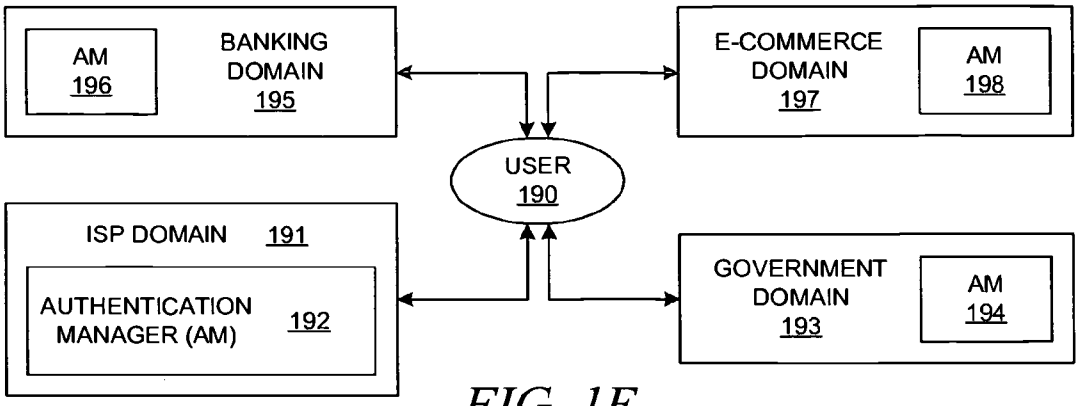


FIG. 1E
(PRIOR ART)

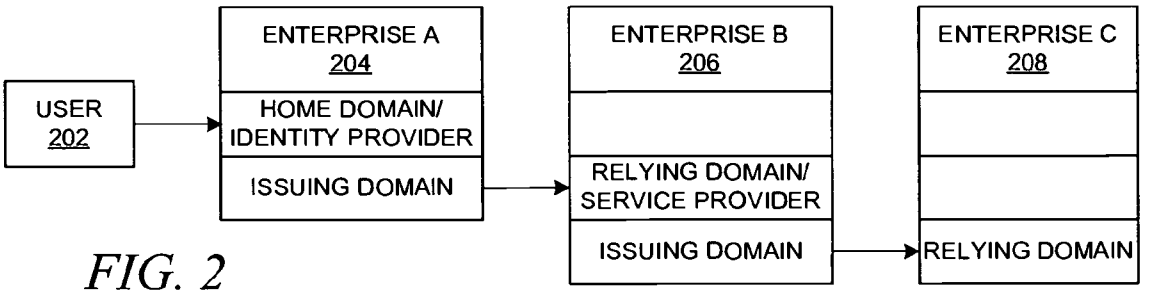


FIG. 2

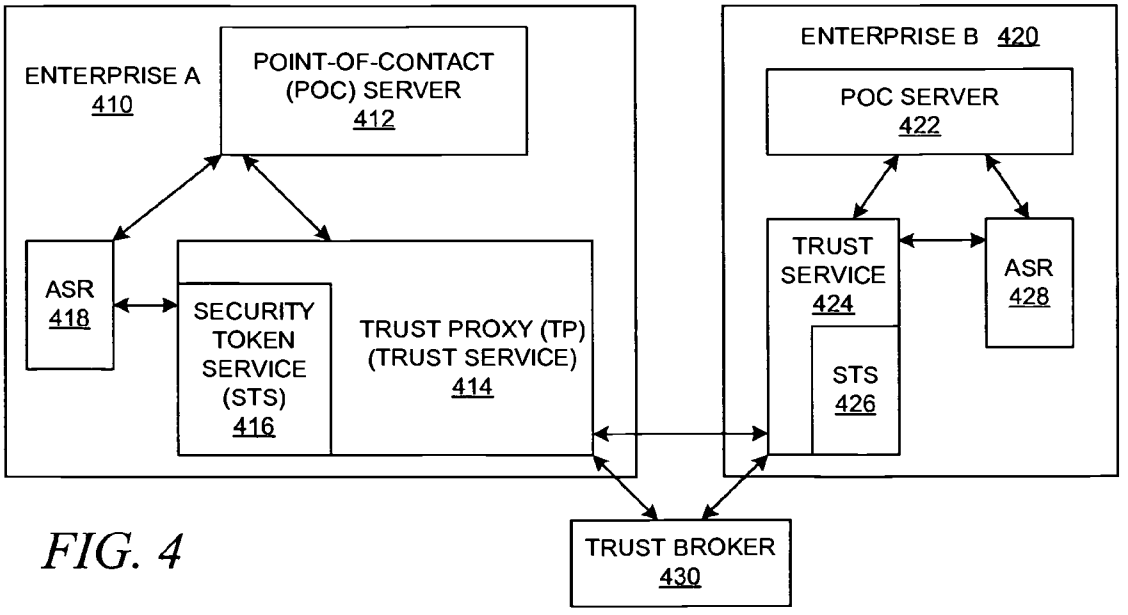
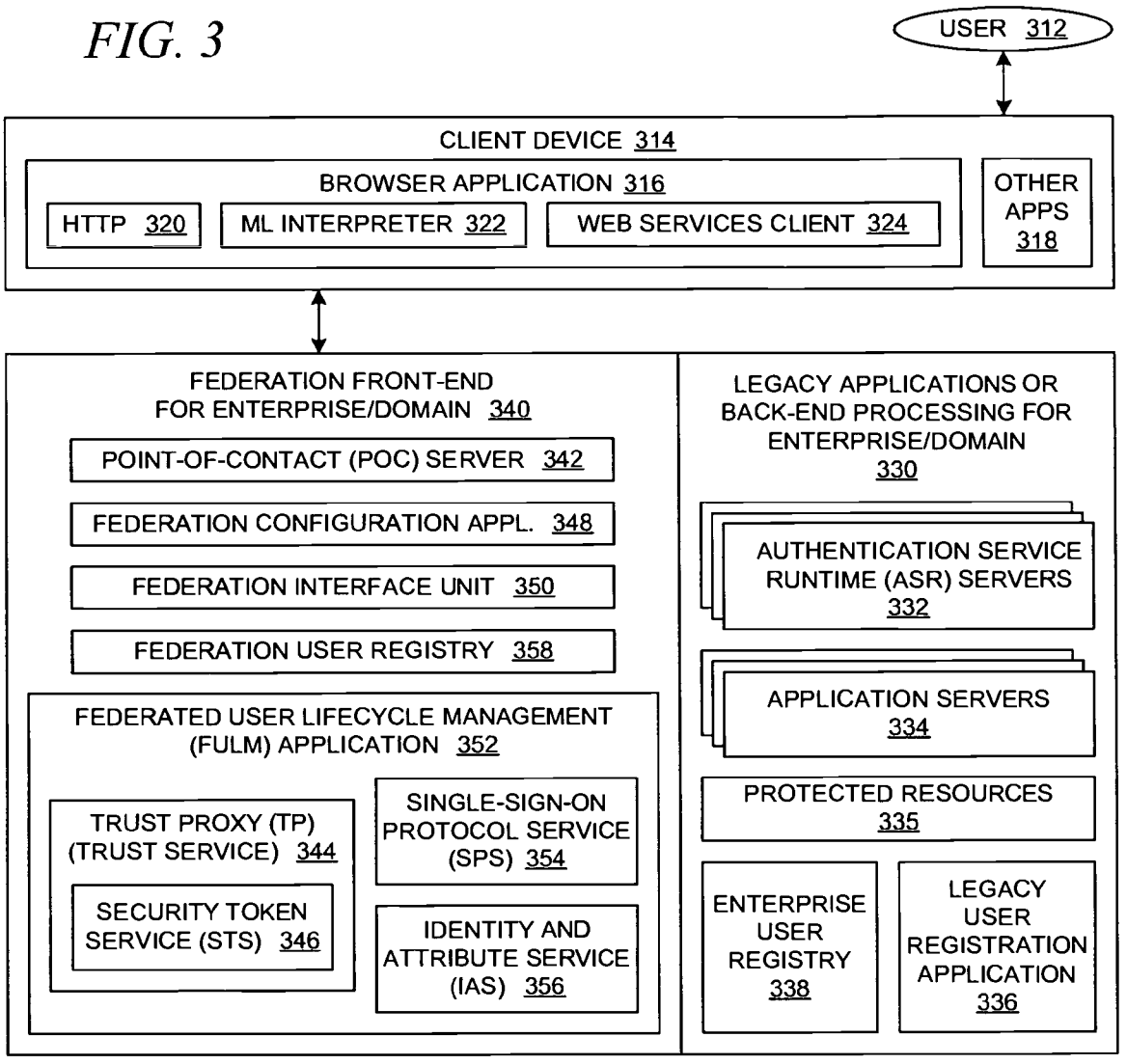


FIG. 4

FIG. 3


USER 312
CLIENT DEVICE 314
BROWSER APPLICATION 316
HTTP 320
ML INTERPRETER 322
WEB SERVICES CLIENT 324
OTHER APPS 318
FEDERATION FRONT-END FOR ENTERPRISE/DOMAIN 340
POINT-OF-CONTACT (POC) SERVER 342
FEDERATION CONFIGURATION APPL. 348
FEDERATION INTERFACE UNIT 350
FEDERATION USER REGISTRY 358
FEDERATED USER LIFECYCLE MANAGEMENT (FULM) APPLICATION 352
TRUST PROXY (TP) (TRUST SERVICE) 344
SECURITY TOKEN SERVICE (STS) 346
SINGLE-SIGN-ON PROTOCOL SERVICE (SPS) 354
IDENTITY AND ATTRIBUTE SERVICE (IAS) 356
LEGACY APPLICATIONS OR BACK-END PROCESSING FOR ENTERPRISE/DOMAIN 330
AUTHENTICATION SERVICE RUNTIME (ASR) SERVERS 332
APPLICATION SERVERS 334
PROTECTED RESOURCES 335
ENTERPRISE USER REGISTRY 338
LEGACY USER REGISTRATION APPLICATION 336

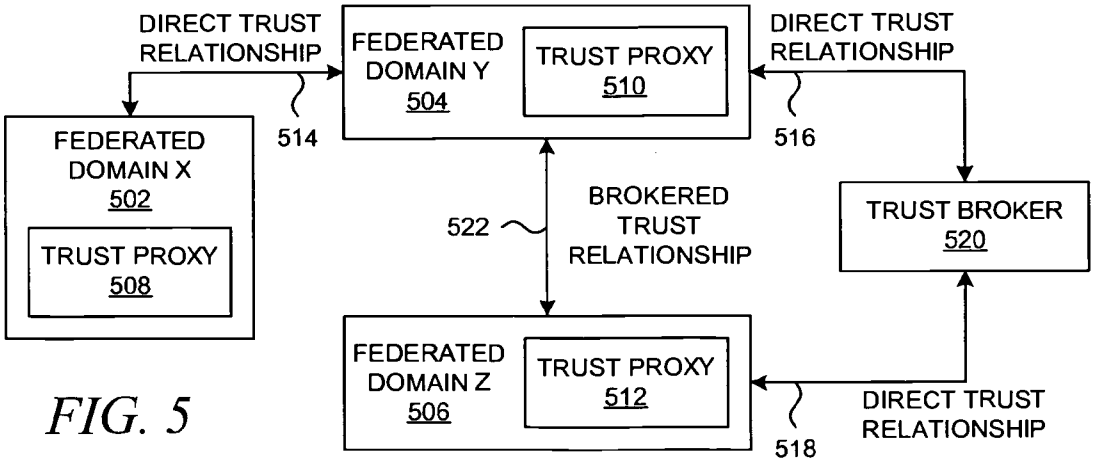

DIRECT TRUST RELATIONSHIP
FEDERATED DOMAIN Y 504
TRUST PROXY 510
DIRECT TRUST RELATIONSHIP
FEDERATED DOMAIN X 502
TRUST PROXY 508
514
516
522
BROKERED TRUST RELATIONSHIP
TRUST BROKER 520
FEDERATED DOMAIN Z 506
TRUST PROXY 512
DIRECT TRUST RELATIONSHIP
518

FIG. 5

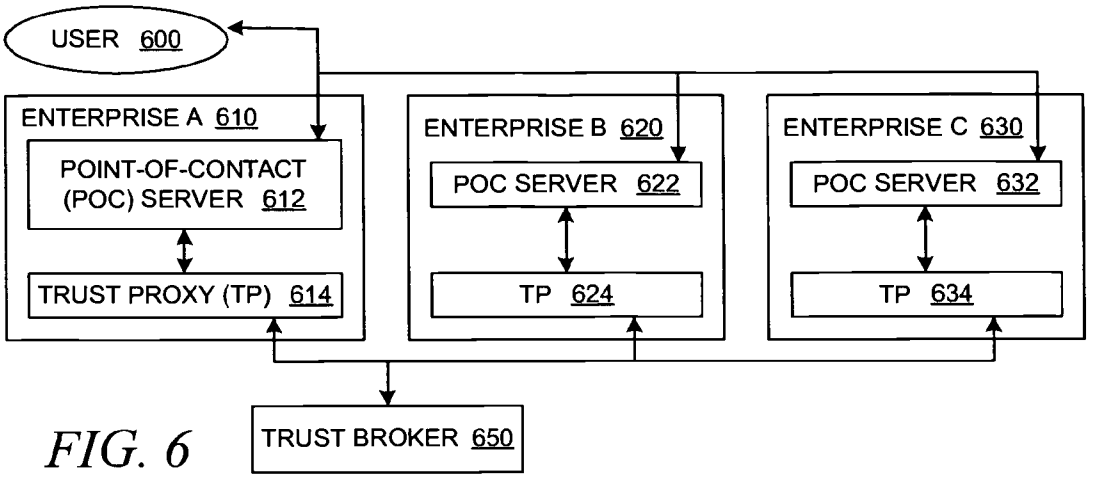

USER 600
ENTERPRISE A 610
POINT-OF-CONTACT (POC) SERVER 612
TRUST PROXY (TP) 614
ENTERPRISE B 620
POC SERVER 622
TP 624
ENTERPRISE C 630
POC SERVER 632
TP 634
TRUST BROKER 650

FIG. 6

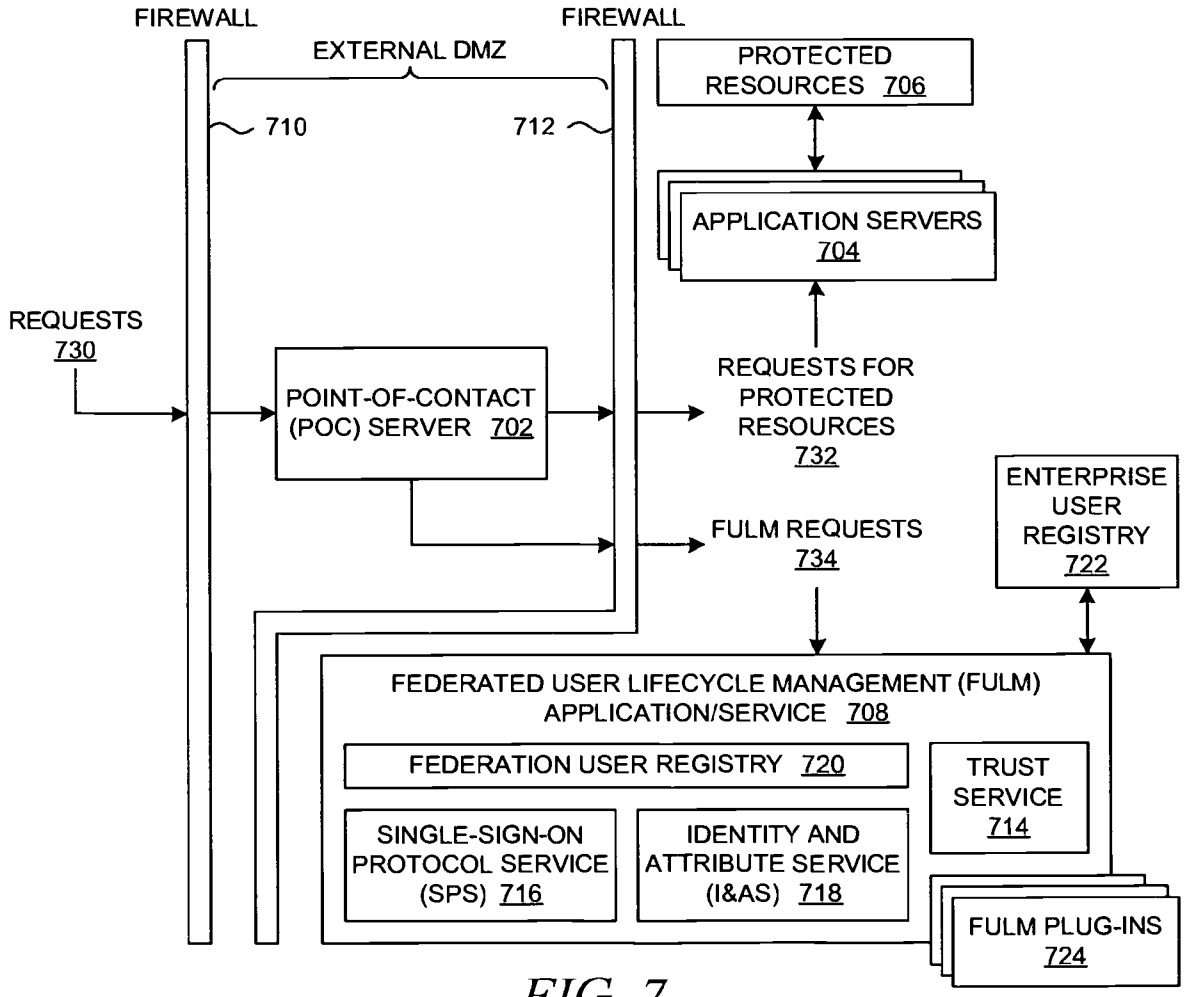
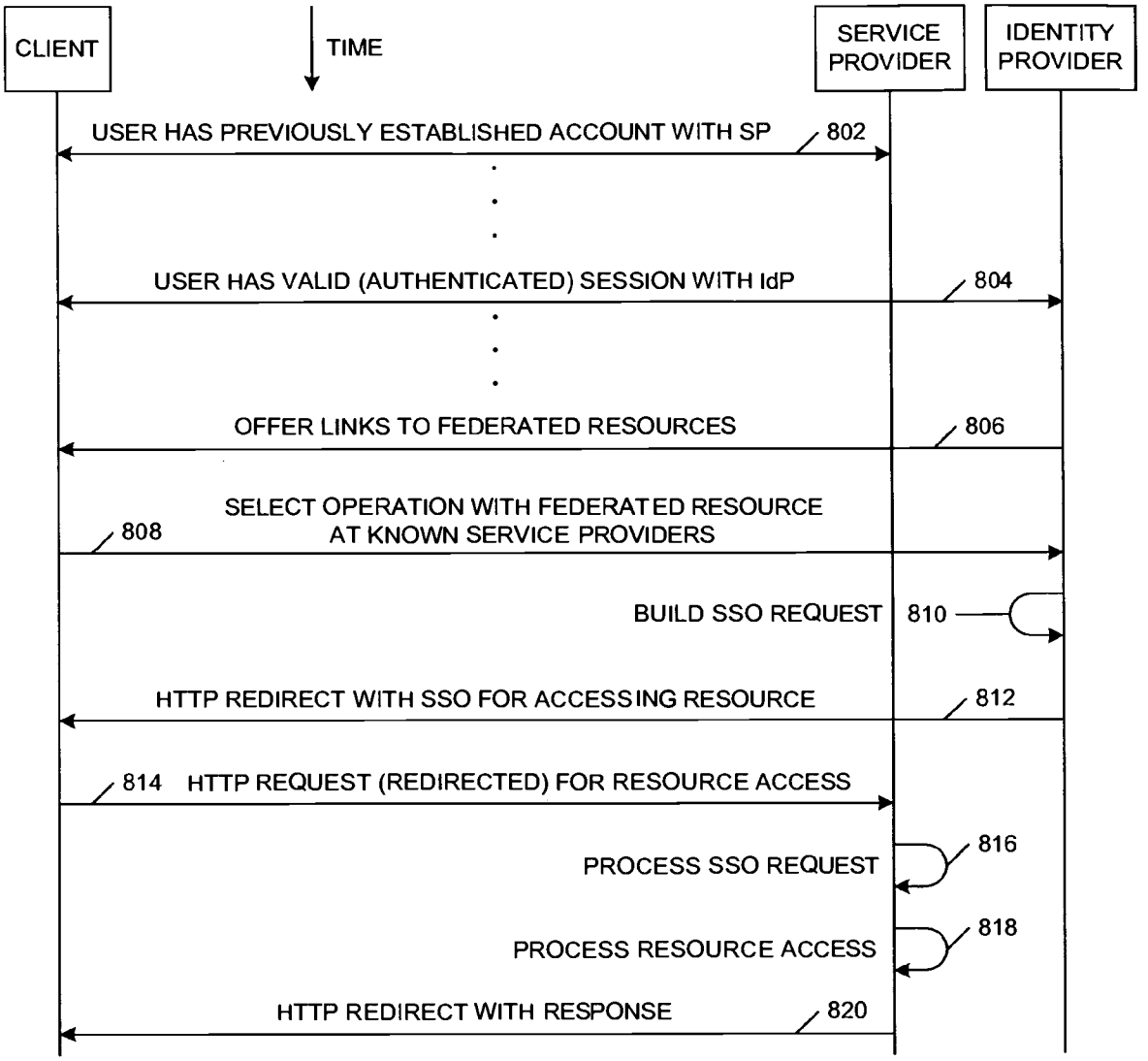
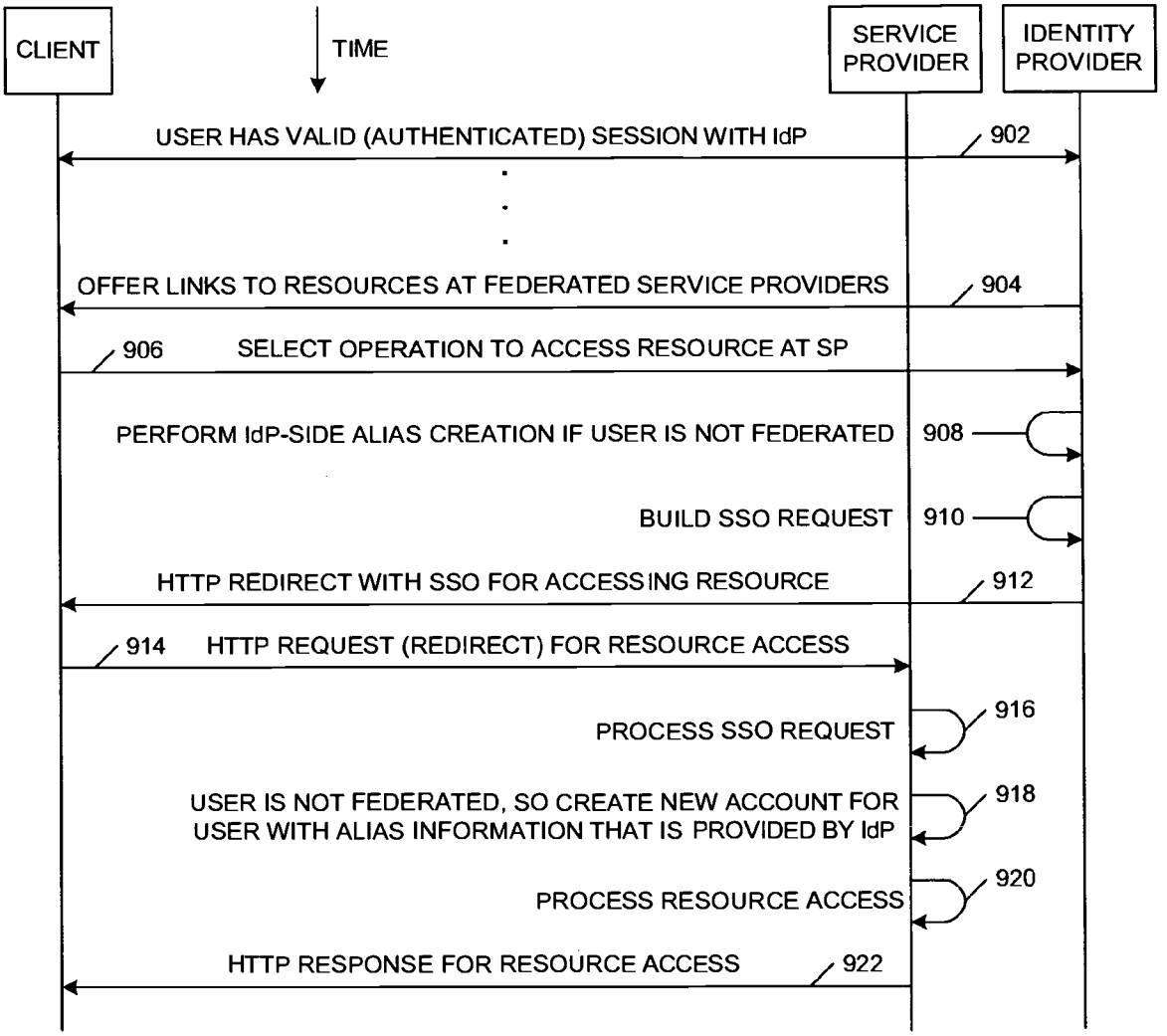

FIREWALL
EXTERNAL DMZ
FIREWALL
710
712
PROTECTED RESOURCES 706
APPLICATION SERVERS 704
REQUESTS 730
POINT-OF-CONTACT (POC) SERVER 702
REQUESTS FOR PROTECTED RESOURCES 732
ENTERPRISE USER REGISTRY 722
FULM REQUESTS 734
FEDERATED USER LIFECYCLE MANAGEMENT (FULM) APPLICATION/SERVICE 708
FEDERATION USER REGISTRY 720
TRUST SERVICE 714
SINGLE-SIGN-ON PROTOCOL SERVICE (SPS) 716
IDENTITY AND ATTRIBUTE SERVICE (I&AS) 718
FULM PLUG-INS 724

FIG. 7



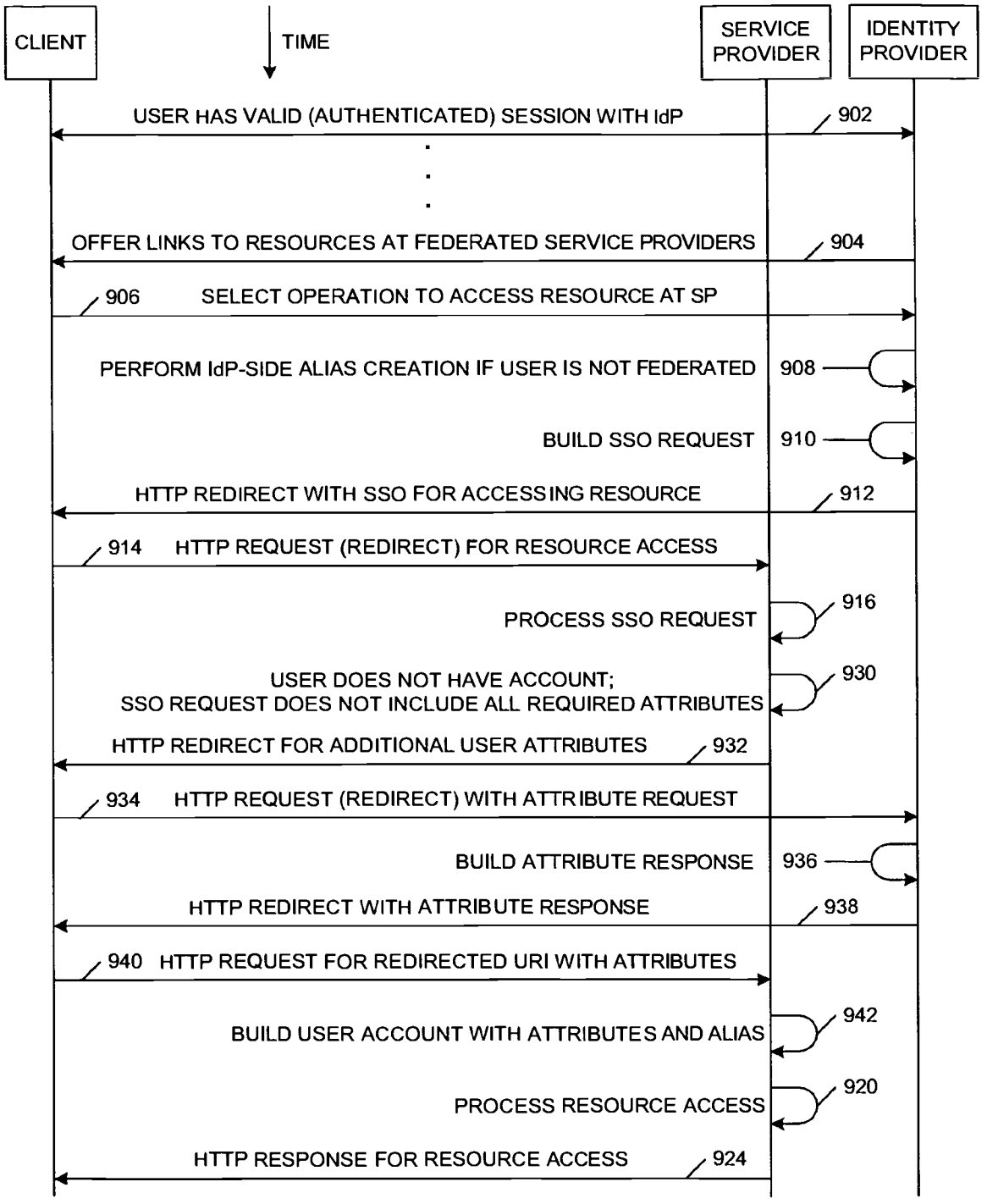
TYPICAL SINGLE-SIGN-ON OPERATION
(INITIATED BY IDENTITY PROVIDER -- USER PREVIOUSLY PROVISIONED AT SP)

FIG. 8
(PRIOR ART)



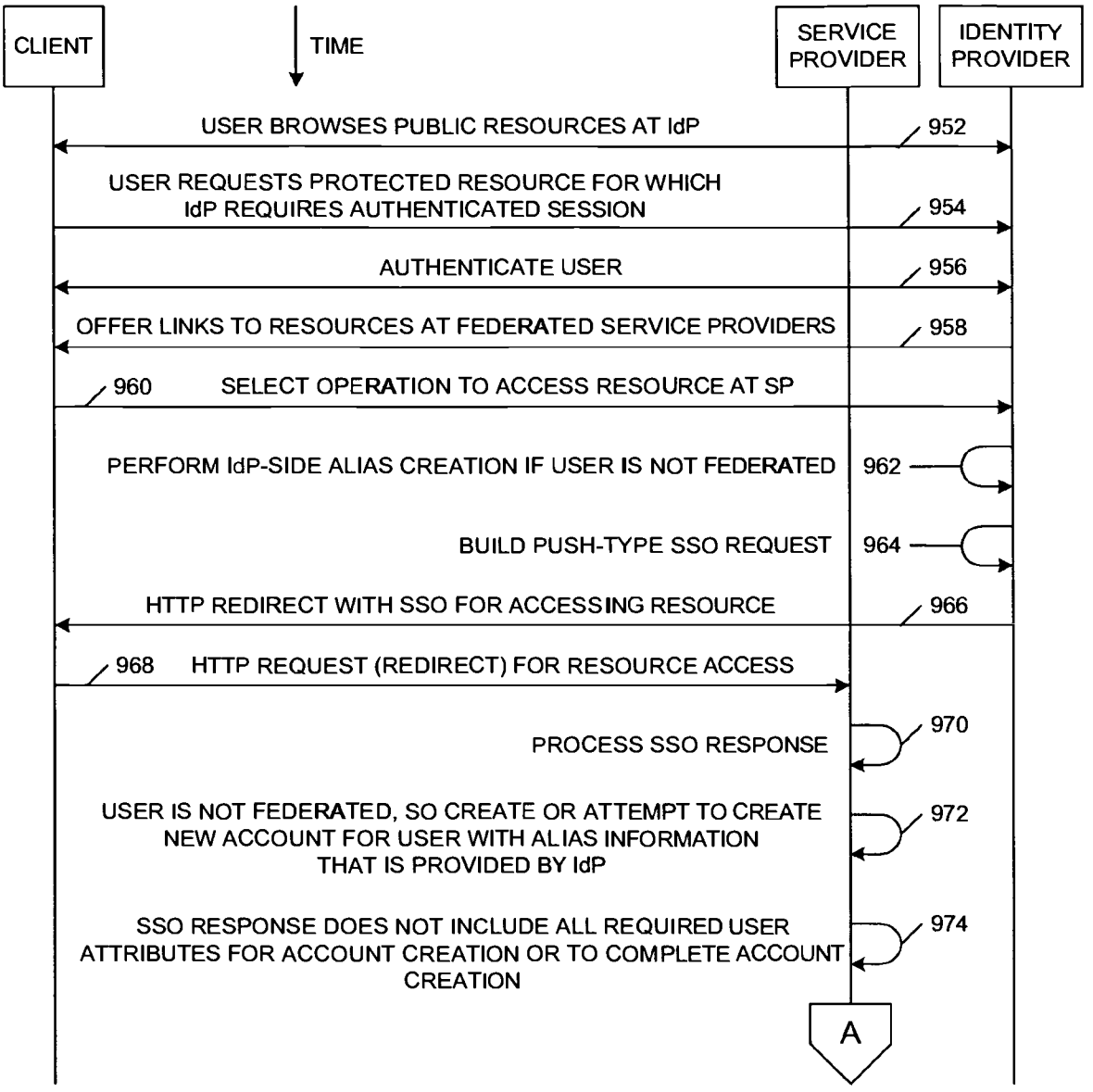
PUSH-TYPE SINGLE-SIGN-ON OPERATION WITH RUNTIME USER ACCOUNT CREATION AT SP (USER NOT PREVIOUSLY PROVISIONED AT SP)

FIG. 9A



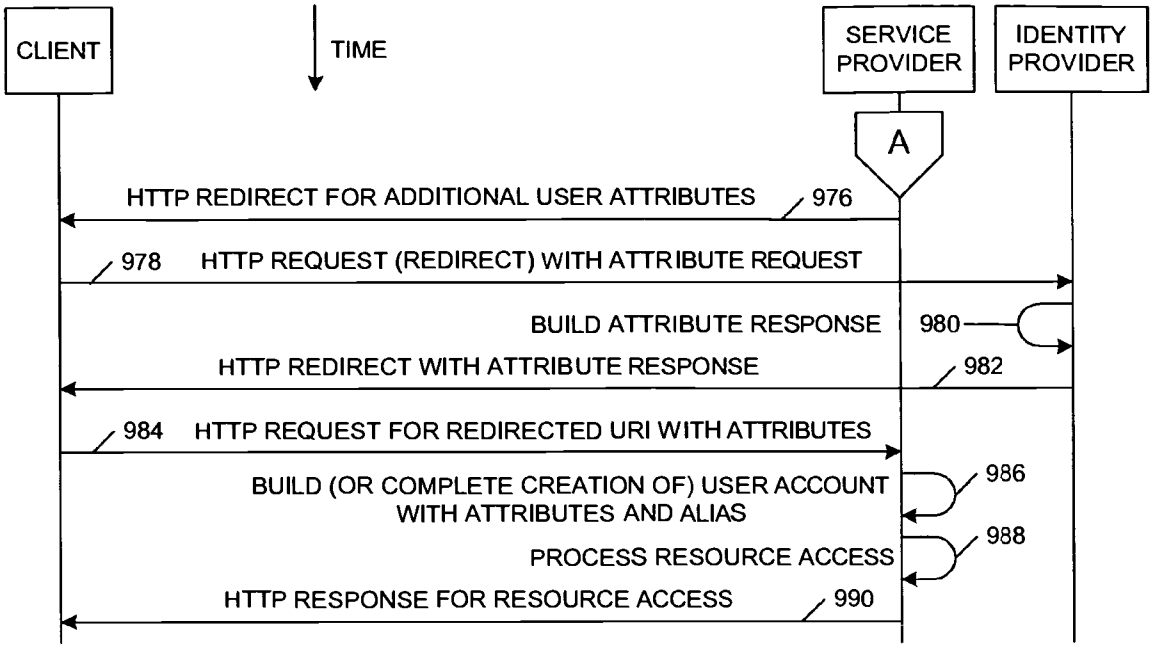
PUSH-TYPE SINGLE-SIGN-ON OPERATION WITH RUNTIME USER ACCOUNT CREATION AT SP (ADDITIONAL PULLING OF USER ATTRIBUTES BY SP FROM IDP)

FIG. 9B



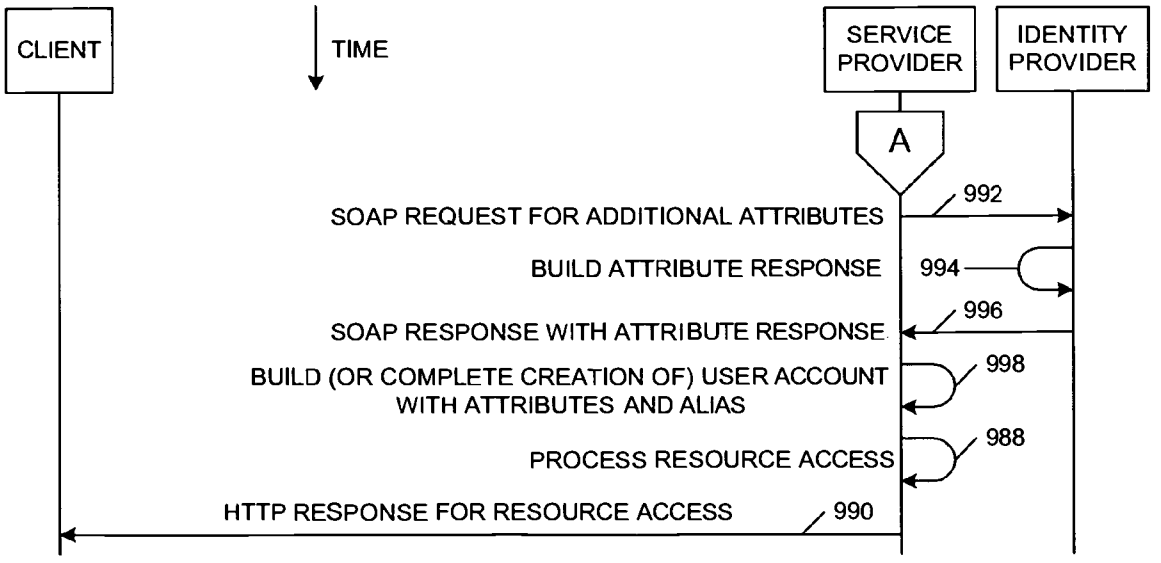
PUSH-TYPE SINGLE-SIGN-ON OPERATION WITH RUNTIME USER ACCOUNT CREATION AT SP (ADDITIONAL PULLING OF USER ATTRIBUTES BY SP FROM IDP)

FIG. 9C



COMPLETION OF PUSH-TYPE SSO OPERATION WITH RUNTIME USER ACCOUNT CREATION AT SP (FRONT-CHANNEL USER ATTRIBUTE RETRIEVAL BY SP FROM IDP)

FIG. 9D



COMPLETION OF PUSH-TYPE SSO OPERATION WITH RUNTIME USER ACCOUNT CREATION AT SP (BACK-CHANNEL USER ATTRIBUTE RETRIEVAL BY SP FROM IDP)

FIG. 9E

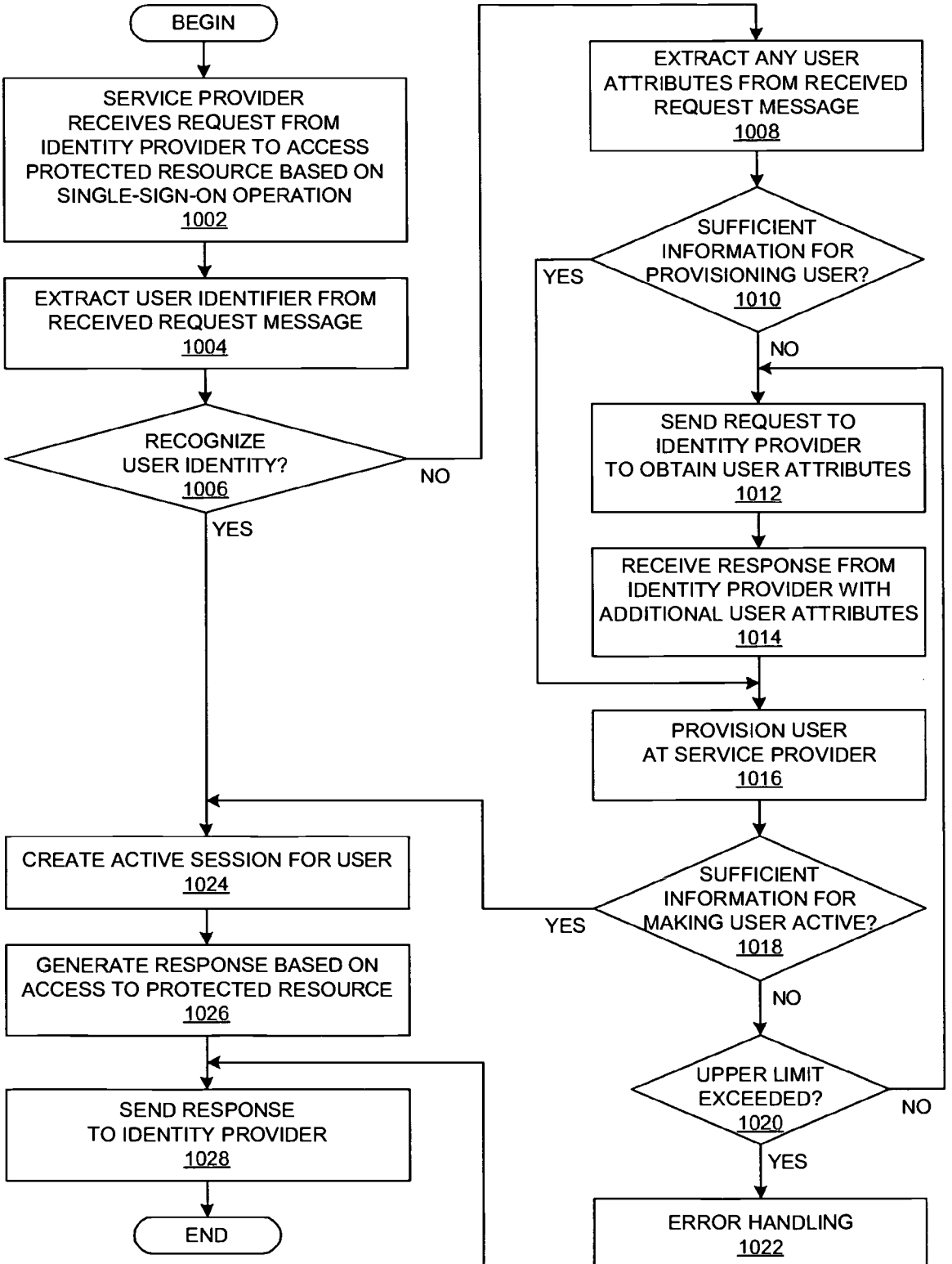
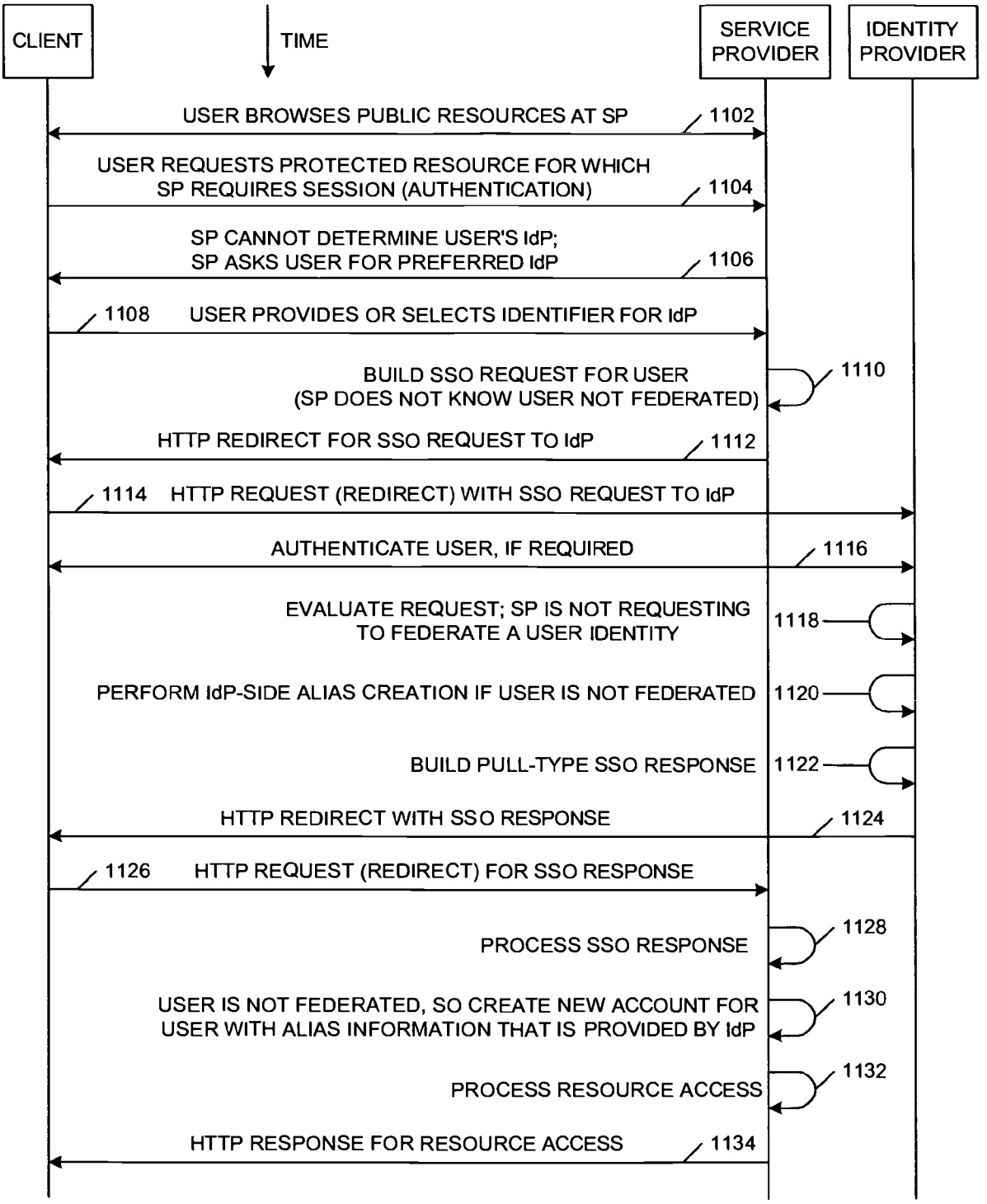
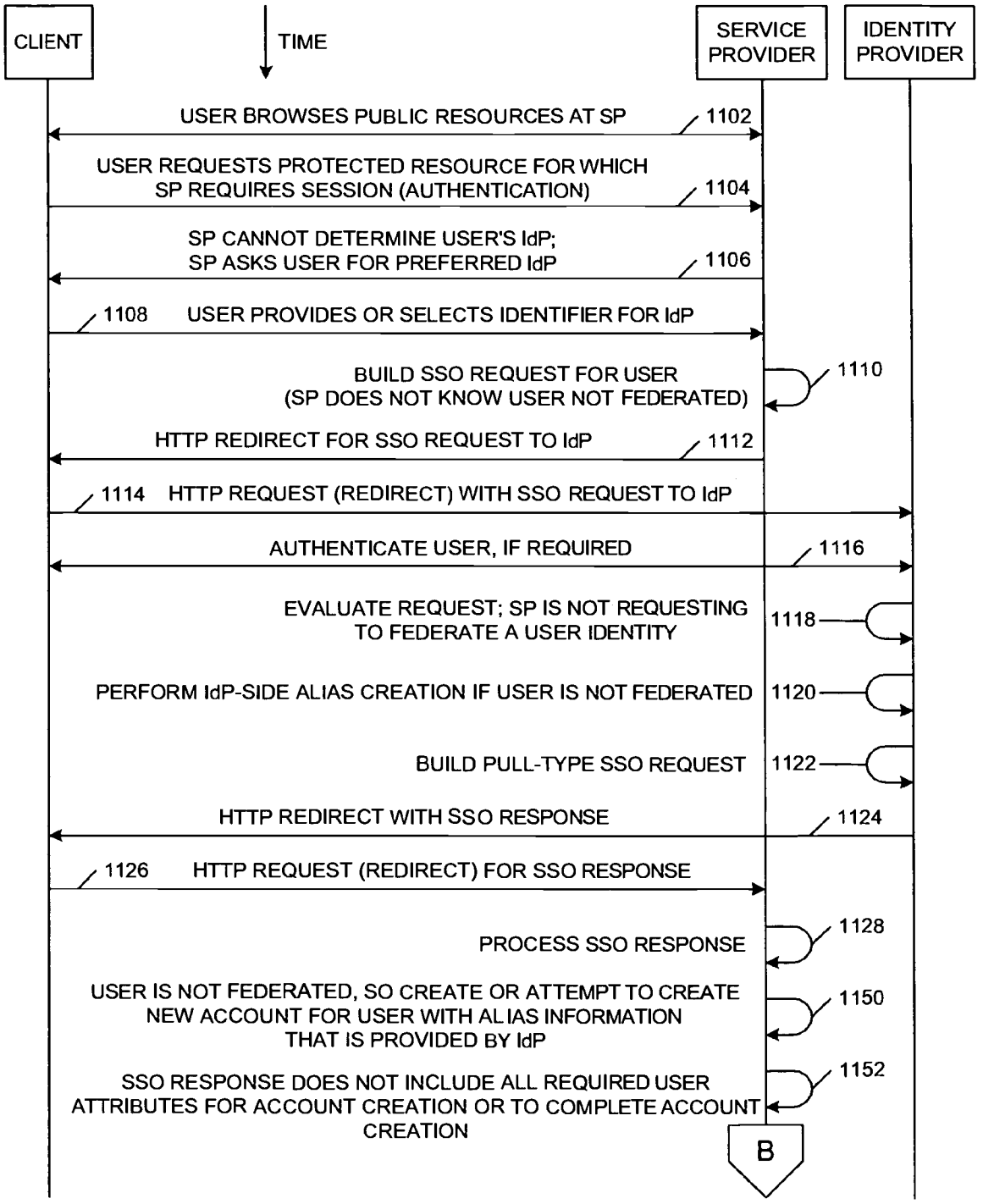

BEGIN
SERVICE PROVIDER RECEIVES REQUEST FROM IDENTITY PROVIDER TO ACCESS PROTECTED RESOURCE BASED ON SINGLE-SIGN-ON OPERATION 1002
EXTRACT USER IDENTIFIER FROM RECEIVED REQUEST MESSAGE 1004
RECOGNIZE USER IDENTITY? 1006
NO
YES
EXTRACT ANY USER ATTRIBUTES FROM RECEIVED REQUEST MESSAGE 1008
SUFFICIENT INFORMATION FOR PROVISIONING USER? 1010
YES
NO
SEND REQUEST TO IDENTITY PROVIDER TO OBTAIN USER ATTRIBUTES 1012
RECEIVE RESPONSE FROM IDENTITY PROVIDER WITH ADDITIONAL USER ATTRIBUTES 1014
PROVISION USER AT SERVICE PROVIDER 1016
SUFFICIENT INFORMATION FOR MAKING USER ACTIVE? 1018
YES
NO
UPPER LIMIT EXCEEDED? 1020
NO
YES
ERROR HANDLING 1022
CREATE ACTIVE SESSION FOR USER 1024
GENERATE RESPONSE BASED ON ACCESS TO PROTECTED RESOURCE 1026
SEND RESPONSE TO IDENTITY PROVIDER 1028
END

FIG. 10



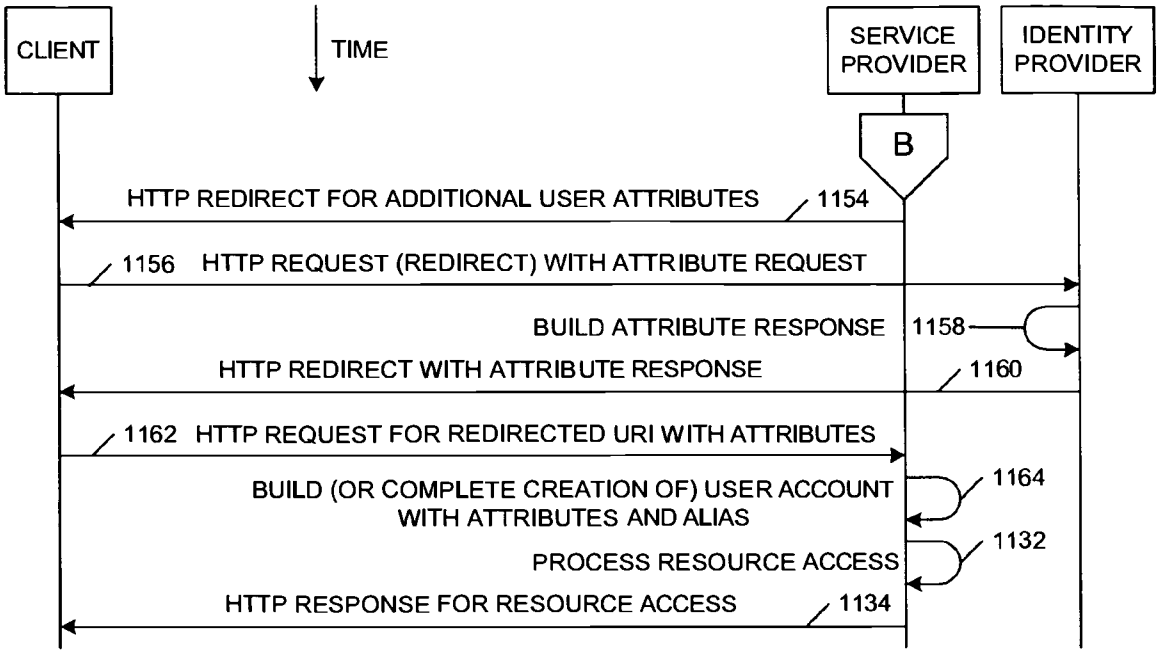
PULL-TYPE SINGLE-SIGN-ON OPERATION WITH RUNTIME USE R PROVISIONING AT SP (USER NOT PREVIOUSLY PROVISIONED AT SP)

FIG. 11A



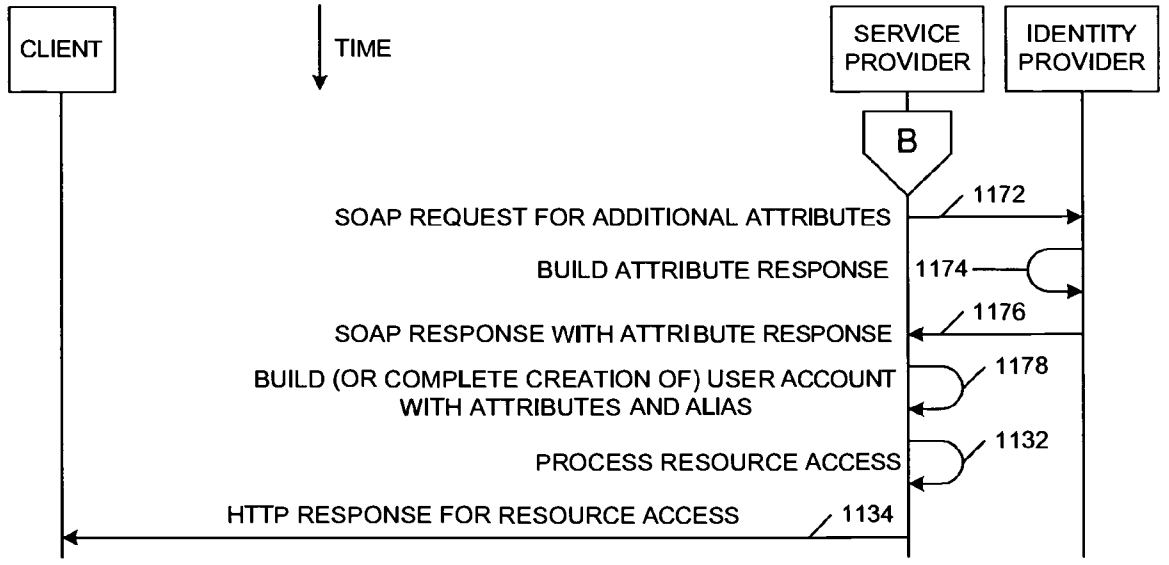
PULL-TYPE SINGLE-SIGN-ON OPERATION WITH RUNTIME USER PROVISIONING AT SP (REQUIRES ADDITIONAL PULLING OF USER ATTRIBUTES BY SP FROM IDP)

FIG. 11B



COMPLETION OF PULL-TYPE SSO OPERATION WITH RUNTIME USER ACCOUNT CREATION AT SP (FRONT-CHANNEL USER ATTRIBUTE RETRIEVAL BY SP FROM IDP)

FIG. 11C



COMPLETION OF PULL-TYPE SSO OPERATION WITH RUNTIME USER ACCOUNT CREATION AT SP (BACK-CHANNEL USER ATTRIBUTE RETRIEVAL BY SP FROM IDP)

FIG. 11D

METHOD AND SYSTEM FOR A RUNTIME USER ACCOUNT CREATION OPERATION WITHIN A SINGLE-SIGN-ON PROCESS IN A FEDERATED COMPUTING ENVIRONMENT

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to an improved data processing system and, in particular, to a method and apparatus for multicomputer data transferring. Still more particularly, the present invention is directed to networked computer systems.

2. Description of Related Art

Enterprises generally desire to provide authorized users with secure access to protected resources in a user-friendly manner throughout a variety of networks, including the Internet. Although providing secure authentication mechanisms reduces the risks of unauthorized access to protected resources, those authentication mechanisms may become barriers to accessing protected resources. Users generally desire the ability to change from interacting with one application to another application without regard to authentication barriers that protect each particular system supporting those applications.

As users get more sophisticated, they expect that computer systems coordinate their actions so that burdens on the user are reduced. These types of expectations also apply to authentication processes. A user might assume that once he or she has been authenticated by some computer system, the authentication should be valid throughout the user's working session, or at least for a particular period of time, without regard to the various computer architecture boundaries that are almost invisible to the user. Enterprises generally try to fulfill these expectations in the operational characteristics of their deployed systems, not only to placate users but also to increase user efficiency, whether the user efficiency is related to employee productivity or customer satisfaction.

More specifically, with the current computing environment in which many applications have a Web-based user interface that is accessible through a common browser, users expect more user-friendliness and low or infrequent barriers to movement from one Web-based application to another. In this context, users are coming to expect the ability to jump from interacting with an application on one Internet domain to another application on another domain without regard to the authentication barriers that protect each particular domain. However, even if many systems provide secure authentication through easy-to-use, Web-based interfaces, a user may still be forced to reckon with multiple authentication processes that stymie user access across a set of domains. Subjecting a user to multiple authentication processes in a given time frame may significantly affect the user's efficiency.

For example, various techniques have been used to reduce authentication burdens on users and computer system administrators. These techniques are generally described as "single-sign-on" (SSO) processes because they have a common purpose: after a user has completed a sign-on operation, i.e. been authenticated, the user is subsequently not required to perform another authentication operation. Hence, the goal is that the user would be required to complete only one authentication process during a particular user session.

To reduce the costs of user management and to improve interoperability among enterprises, federated computing spaces have been created. A federation is a loosely coupled affiliation of enterprises which adhere to certain standards of interoperability; the federation provides a mechanism for trust among those enterprises with respect to certain computational operations for the users within the federation. For example, a federation partner may act as a user's home domain or identity provider. Other partners within the same federation may rely on the user's identity provider for primary management of the user's authentication credentials, e.g., accepting a single-sign-on token that is provided by the user's identity provider.

As enterprises move to support federated business interactions, these enterprises should provide a user experience that reflects the increased cooperation between two businesses. As noted above, a user may authenticate to one party that acts as an identity provider and then single-sign-on to a federated business partner that acts as a service provider. In conjunction with single-sign-on functionality, additional user lifecycle functionality, such as single-sign-off, user provisioning, and account linking/delinking, should also be supported.

Single-sign-on solutions require that a user be identifiable in some form or another at both an identity provider and a service provider; the identity provider needs to be able to identify and authenticate a user, and the service provider needs to be able to identify the user based on some form of assertion about the user in response to a single-sign-on request. Various prior art single-sign-on solutions, e.g., such as those described in the Liberty Alliance ID-FF specifications, require that a user have an authenticatable account at both an identity provider and a service provider as a prerequisite to a federated single-sign-on operation. Some federated solutions support an a priori user account creation event across domains to be used to establish these accounts, thereby satisfying a requirement that a user have an authenticatable account at both an identity provider and a service provider as a prerequisite to a federated single-sign-on operation. Although some federated solutions provide a robust set of federated user lifecycle management operations, such as user account creation, user account management, user attribute management, account suspension, and account deletion, these federated management systems do not provide a lightweight solution that is suitable for certain federation partners or for certain federated purposes.

Therefore, it would be advantageous to have methods and systems in which enterprises can provide comprehensive single-sign-on experiences to users in a federated computing environment in a lightweight manner that does not require an extensive amount of a priori processing.

SUMMARY OF THE INVENTION

A method, system, apparatus, and computer program product are presented to support computing systems of different enterprises that interact within a federated computing environment. Federated single-sign-on operations can be initiated at the computing systems of federation partners on behalf of a user even though the user has not established a user account at a federation partner prior to the initiation of the single-sign-on operation. For example, an identity provider can initiate a single-sign-on operation at a service provider while attempting to obtain access to a controlled resource on behalf of a user. When the service provider recognizes that it does not have a linked user account for the user that allows a single-sign-on operation from the identity provider, the service provider creates a local user account based at least in part on information from the identity provider. The service provider

can also pull user attributes from the identity provider as necessary to perform the user account creation operation.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, further objectives, and advantages thereof, will be best understood by reference to the following detailed description when read in conjunction with the accompanying drawings, wherein:

FIG. **1**A depicts a typical network of data processing systems, each of which may implement the present invention;

FIG. **1**B depicts a typical computer architecture that may be used within a data processing system in which the present invention may be implemented;

FIG. **1**C depicts a data flow diagram that illustrates a typical authentication process that may be used when a client attempts to access a protected resource at a server;

FIG. **1**D depicts a network diagram that illustrates a typical Web-based environment in which the present invention may be implemented;

FIG. **1**E depicts a block diagram that illustrates an example of a typical online transaction that might require multiple authentication operations from a user;

FIG. **2** depicts a block diagram that illustrates the terminology of the federated environment with respect to a transaction that is initiated by a user to a first federated enterprise, which, in response, invokes actions at downstream entities within the federated environment;

FIG. **3** depicts a block diagram that illustrates the integration of pre-existing data processing systems at a given domain with some federated architecture components that may be used to support an embodiment of the present invention;

FIG. **4** depicts a block diagram that illustrates an example of a manner in which some components within a federated architecture may be used to establish trust relationships to support an implementation of the present invention;

FIG. **5** depicts a block diagram that illustrates an exemplary set of trust relationships between federated domains using trust proxies and a trust broker in accordance with an exemplary federated architecture that is able to support the present invention;

FIG. **6** depicts a block diagram that illustrates a federated environment that supports federated single-sign-on operations;

FIG. **7** depicts a block diagram that illustrates some of the components in a federated domain for implementing federated user lifecycle management functionality in order to support the present invention;

FIG. **8** depicts a dataflow diagram that shows a typical prior art HTTP-redirection-based single-sign-on operation that is initiated by a federated identity provider to obtain access to a protected resource at a federated service provider;

FIGS. **9**A-**9**B depicts dataflow diagrams that show an HTTP-redirection-based single-sign-on operation that is initiated by a federated identity provider to obtain access to a protected resource at a federated service provider while performing a runtime linked-user-account creation operation at the federated service provider in accordance with an embodiment of the present invention;

FIGS. **9**C-**9**E depict dataflow diagrams that show an HTTP-redirection-based single-sign-on operation that is initiated by a federated identity provider to obtain access to a protected resource at a federated service provider with alternative methods for obtaining user attributes by the federated service provider in accordance with an embodiment of the present invention;

FIG. **10** depicts a flowchart that shows a more detailed process for performing a runtime linked-user-account creation operation at a service provider during a single-sign-on operation that has been initiated by an identity provider;

FIG. **11**A depicts a dataflow diagram that shows an HTTP-redirection-based pull-type single-sign-on operation that is initiated by a federated service provider to allow access to a protected resource at the federated service provider while performing a runtime linked-user-account creation operation at the federated service provider in accordance with an embodiment of the present invention; and

FIGS. **11**B-**11**D depict a set of dataflow diagrams that show an HTTP-redirection-based pull-type single-sign-on operation that is initiated by a federated service provider to allow access to a protected resource at the federated service provider with additional retrieval of user attribute information from a federated identity provider while performing a runtime linked-user-account creation operation at the federated service provider in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

In general, the devices that may comprise or relate to the present invention include a wide variety of data processing technology. Therefore, as background, a typical organization of hardware and software components within a distributed data processing system is described prior to describing the present invention in more detail.

With reference now to the figures, FIG. **1**A depicts a typical network of data processing systems, each of which may implement the present invention. Distributed data processing system **100** contains network **101**, which is a medium that may be used to provide communications links between various devices and computers connected together within distributed data processing system **100**. Network **101** may include permanent connections, such as wire or fiber optic cables, or temporary connections made through telephone or wireless communications. In the depicted example, server **102** and server **103** are connected to network **101** along with storage unit **104**. In addition, clients **105**-**107** also are connected to network **101**. Clients **105**-**107** and servers **102**-**103** may be represented by a variety of computing devices, such as mainframes, personal computers, personal digital assistants (PDAs), etc. Distributed data processing system **100** may include additional servers, clients, routers, other devices, and peer-to-peer architectures that are not shown.

In the depicted example, distributed data processing system **100** may include the Internet with network **101** representing a worldwide collection of networks and gateways that use various protocols to communicate with one another, such as LDAP (Lightweight Directory Access Protocol), TCP/IP (Transport Control Protocol/Internet Protocol), HTTP (HyperText Transport Protocol), etc. Of course, distributed data processing system **100** may also include a number of different types of networks, such as, for example, an intranet, a local area network (LAN), or a wide area network (WAN). For example, server **102** directly supports client **109** and network **110**, which incorporates wireless communication links. Network-enabled phone **111** connects to network **110** through wireless link **112**, and PDA **113** connects to network **110** through wireless link **114**. Phone **111** and PDA **113** can also directly transfer data between themselves across wireless link **115** using an appropriate technology, such as Bluetooth™

wireless technology, to create so-called personal area networks or personal ad-hoc networks. In a similar manner, PDA **113** can transfer data to PDA **107** via wireless communication link **116**.

The present invention could be implemented on a variety of hardware platforms and software environments. FIG. **1**A is intended as an example of a heterogeneous computing environment and not as an architectural limitation for the present invention.

With reference now to FIG. **1**B, a diagram depicts a typical computer architecture of a data processing system, such as those shown in FIG. **1**A, in which the present invention may be implemented. Data processing system **120** contains one or more central processing units (CPUs) **122** connected to internal system bus **123**, which interconnects random access memory (RAM) **124**, read-only memory **126**, and input/output adapter **128**, which supports various I/O devices, such as printer **130**, disk units **132**, or other devices not shown, such as a audio output system, etc. System bus **123** also connects communication adapter **134** that provides access to communication link **136**. User interface adapter **148** connects various user devices, such as keyboard **140** and mouse **142**, or other devices not shown, such as a touch screen, stylus, microphone, etc. Display adapter **144** connects system bus **123** to display device **146**.

Those of ordinary skill in the art will appreciate that the hardware in FIG. **1**B may vary depending on the system implementation. For example, the system may have one or more processors, such as an Intel® Pentium®-based processor and a digital signal processor (DSP), and one or more types of volatile and non-volatile memory. Other peripheral devices may be used in addition to or in place of the hardware depicted in FIG. **1**B. The depicted examples are not meant to imply architectural limitations with respect to the present invention.

In addition to being able to be implemented on a variety of hardware platforms, the present invention may be implemented in a variety of software environments. A typical operating system may be used to control program execution within each data processing system. For example, one device may run a Unix® operating system, while another device contains a simple Java® runtime environment. A representative computer platform may include a browser, which is a well known software application for accessing hypertext documents in a variety of formats, such as graphic files, word processing files, Extensible Markup Language (XML), Hypertext Markup Language (HTML), Handheld Device Markup Language (HDML), Wireless Markup Language (WML), and various other formats and types of files. It should also be noted that the distributed data processing system shown in FIG. **1**A is contemplated as being fully able to support a variety of peer-to-peer subnets and peer-to-peer services.

With reference now to FIG. **1**C, a data flow diagram illustrates a typical authentication process that may be used when a client attempts to access a protected resource at a server. As illustrated, the user at a client workstation **150** seeks access over a computer network to a protected resource on a server **151** through the user's web browser executing on the client workstation. A protected or controlled resource is a resource (an application, an object, a document, a page, a file, executable code, or other computational resource, communication-type resource, etc.) for which access is controlled or restricted. A protected resource is identified by a Uniform Resource Locator (URL), or more generally, a Uniform Resource Identifier (URI), that can only be accessed by an authenticated and/or authorized user. The computer network may be the Internet, an intranet, or other network, as shown in FIG. **1**A or FIG. **1**B, and the server may be a web application server (WAS), a server application, a servlet process, or the like.

The process is initiated when the user requests a server-side protected resource, such as a web page within the domain "ibm.com" (step **152**). The terms "server-side" and "client-side" refer to actions or entities at a server or a client, respectively, within a networked environment. The web browser (or associated application or applet) generates an HTTP request (step **153**) that is sent to the web server that is hosting the domain "ibm.com". The terms "request" and "response" should be understood to comprise data formatting that is appropriate for the transfer of information that is involved in a particular operation, such as messages, communication protocol information, or other associated information.

The server determines that it does not have an active session for the client (step **154**), so the server initiates and completes the establishment of an SSL (Secure Sockets Layer) session between the server and the client (step **155**), which entails multiple transfers of information between the client and the server. After an SSL session is established, subsequent communication messages are transferred within the SSL session; any secret information remains secure because of the encrypted communications within the SSL session.

However, the server needs to determine the identity of the user before allowing the user to have access to protected resources, so the server requires the user to perform an authentication process by sending the client some type of authentication challenge (step **156**). The authentication challenge may be in various formats, such as an HTML form. The user then provides the requested or required information (step **157**), such as a username or other type of user identifier along with an associated password or other form of secret information.

The authentication response information is sent to the server (step **158**), at which point the server authenticates the user or client (step **159**), e.g., by retrieving previously submitted registration information and matching the presented authentication information with the user's stored information. Assuming the authentication is successful, an active session is established for the authenticated user or client. The server creates a session identifier for the client, and any subsequent request messages from the client within the session would be accompanied by the session identifier.

The server then retrieves the originally requested web page and sends an HTTP response message to the client (step **160**), thereby fulfilling the user's original request for the protected resource. At that point, the user may request another page within "ibm.com" (step **161**) by clicking a hypertext link within a browser window, and the browser sends another HTTP request message to the server (step **162**). At that point, the server recognizes that the user has an active session (step **163**) because the user's session identifier is returned to the server in the HTTP request message, and the server sends the requested web page back to the client in another HTTP response message (step **164**). Although FIG. **1**C depicts a typical prior art process, it should be noted that other alternative session state management techniques may be depicted, such as URL rewriting or using cookies to identify users with active sessions, which may include using the same cookie that is used to provide proof of authentication.

With reference now to FIG. **1**D, a diagram illustrates a typical Web-based environment in which the present invention may be implemented. In this environment, a user of browser **170** at client **171** desires to access a protected

resource on web application server **172** in DNS domain **173**, or on web application server **174** in DNS domain **175**.

In a manner similar to that shown in FIG. **1**C, a user can request a protected resource at one of many domains. In contrast to FIG. **1**C, which shows only a single server at a particular domain, each domain in FIG. **1**D has multiple servers. In particular, each domain may have an associated authentication server **176** and **177**.

In this example, after client **171** issues a request for a protected resource at domain **173**, web application server **172** determines that it does not have an active session for client **171**, and it requests that authentication server **176** perform an appropriate authentication operation with client **171**. Authentication server **176** communicates the result of the authentication operation to web application server **172**. If the user (or browser **170** or client **171** on behalf of the user) is successfully authenticated, then web application server **172** establishes a session for client **171** and returns the requested protected resource. Typically, once the user is authenticated by the authentication server, a cookie may be set and stored in a cookie cache in the browser. FIG. **1**D is merely an example of one manner in which the processing resources of a domain may be shared amongst multiple servers, particularly to perform authentication operations.

In a similar manner, after client **171** issues a request for a protected resource at domain **175**, authentication server **177** performs an appropriate authentication operation with client **171**, after which web application server **174** establishes a session for client **171** and returns the requested protected resource. Hence, FIG. **1**D illustrates that client **171** may have multiple concurrent sessions in different domains yet is required to complete multiple authentication operations to establish those concurrent sessions.

With reference now to FIG. **1**E, a block diagram depicts an example of a typical online transaction that might require multiple authentication operations from a user. Referring again to FIG. **1**C and FIG. **1**D, a user may be required to complete an authentication operation prior to gaining access to a controlled resource, as shown in FIG. **1**C. Although not shown in FIG. **1**C, an authentication manager may be deployed on server **151** to retrieve and employ user information that is required to authenticate a user. As shown in FIG. **1**D, a user may have multiple current sessions within different domains **173** and **175**, and although they are not shown in FIG. **1**D, each domain may employ an authentication manager in place of or in addition to the authentication servers. In a similar manner, FIG. **1**E also depicts a set of domains, each of which support some type of authentication manager. FIG. **1**E illustrates some of the difficulties that a user may experience when accessing multiple domains that require the user to complete an authentication operation for each domain.

User **190** may be registered at ISP domain **191**, which may support authentication manager **192** that authenticates user **190** for the purpose of completing transactions with respect to domain **191**. ISP domain **191** may be an Internet Service Provider (ISP) that provides Internet connection services, email services, and possibly other e-commerce services. Alternatively, ISP domain **191** may be an Internet portal that is frequently accessed by user **190**.

Similarly, domains **193**, **195**, and **197** represent typical web service providers. Government domain **193** supports authentication manager **194** that authenticates users for completing various government-related transactions. Banking domain **195** supports authentication manager **196** that authenticates users for completing transactions with an online bank. E-commerce domain **197** supports authentication manager **198** that authenticates users for completing online purchases.

As noted previously, when a user attempts to move from one domain to another domain within the Internet or World Wide Web by accessing resources at the different domains, a user may be subjected to multiple user authentication requests or requirements, which can significantly slow the user's progress across a set of domains. Using FIG. **1**E as an exemplary environment, user **190** may be involved in a complicated online transaction with e-commerce domain **197** in which the user is attempting to purchase an on-line service that is limited to users who are at least 18 years old and who have a valid driver license, a valid credit card, and a U.S. bank account. This online transaction may involve domains **191**, **193**, **195**, and **197**.

Typically, a user might not maintain an identity and/or attributes within each domain that participates in a typical online transaction. In this example, user **190** may have registered his or her identity with the user's ISP, but to complete the online transaction, the user might also be required to authenticate to domains **193**, **195**, and **197**. If each of the domains does not maintain an identity for the user, then the user's online transaction may fail. Even if the user can be authenticated by each domain, it is not guaranteed that the different domains can transfer information between themselves in order to complete the user's transaction.

Given the preceding brief description of some current technology, the description of the remaining figures relates to federated computer environments in which the present invention may operate. Prior to discussing the present invention in more detail, however, some terminology is introduced.

Terminology

The terms "entity" or "party" generally refers to an organization, an individual, or a system that operates on behalf of an organization, an individual, or another system. The term "domain" connotes additional characteristics within a network environment, but the terms "entity", "party", and "domain" can be used interchangeably. For example, the term "domain" may also refer to a DNS (Domain Name System) domain, or more generally, to a data processing system that includes various devices and applications that appear as a logical unit to exterior entities.

The terms "request" and "response" should be understood to comprise data formatting that is appropriate for the transfer of information that is involved in a particular operation, such as messages, communication protocol information, or other associated information. A protected resource is a resource (an application, an object, a document, a page, a file, executable code, or other computational resource, communication-type resource, etc.) for which access is controlled or restricted.

A token provides direct evidence of a successful operation and is produced by the entity that performs the operation, e.g., an authentication token that is generated after a successful authentication operation. A Kerberos token is one example of an authentication token that may be used with the present invention. More information on Kerberos may be found in Kohl et al., "The Kerberos Network Authentication Service (V5)", Internet Engineering Task Force (IETF) Request for Comments (RFC) 1510, 09/1993.

An assertion provides indirect evidence of some action. Assertions may provide indirect evidence of identity, authentication, attributes, authorization decisions, or other information and/or operations. An authentication assertion provides indirect evidence of authentication by an entity that is not the authentication service but that listened to the authentication service.

A Security Assertion Markup Language (SAML) assertion is an example of a possible assertion format that may be used with the present invention. SAML has been promulgated by

the Organization for the Advancement of Structured Information Standards (OASIS), which is a non-profit, global consortium. SAML is described in "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)", Committee Specification 01, May 31, 2002, as follows:

> The Security Assertion Markup Language (SAML) is an XML-based framework for exchanging security information. This security information is expressed in the form of assertions about subjects, where a subject is an entity (either human or computer) that has an identity in some security domain. A typical example of a subject is a person, identified by his or her email address in a particular Internet DNS domain. Assertions can convey information about authentication acts performed by subjects, attributes of subjects, and authorization decisions about whether subjects are allowed to access certain resources. Assertions are represented as XML constructs and have a nested structure, whereby a single assertion might contain several different internal statements about authentication, authorization, and attributes. Note that assertions containing authentication statements merely describe acts of authentication that happened previously. Assertions are issued by SAML authorities, namely, authentication authorities, attribute authorities, and policy decision points. SAML defines a protocol by which clients can request assertions from SAML authorities and get a response from them. This protocol, consisting of XML-based request and response message formats, can be bound to many different underlying communications and transport protocols; SAML currently defines one binding, to SOAP over HTTP. SAML authorities can use various sources of information, such as external policy stores and assertions that were received as input in requests, in creating their responses. Thus, while clients always consume assertions, SAML authorities can be both producers and consumers of assertions.

The SAML specification states that an assertion is a package of information that supplies one or more statements made by an issuer. SAML allows issuers to make three different kinds of assertion statements: authentication, in which the specified subject was authenticated by a particular means at a particular time; authorization, in which a request to allow the specified subject to access the specified resource has been granted or denied; and attribute, in which the specified subject is associated with the supplied attributes. As discussed further below, various assertion formats can be translated to other assertion formats when necessary.

Authentication is the process of validating a set of credentials that are provided by a user or on behalf of a user. Authentication is accomplished by verifying something that a user knows, something that a user has, or something that the user is, i.e. some physical characteristic about the user. Something that a user knows may include a shared secret, such as a user's password, or by verifying something that is known only to a particular user, such as a user's cryptographic key. Something that a user has may include a smartcard or hardware token. Some physical characteristic about the user might include a biometric input, such as a fingerprint or a retinal map.

An authentication credential is a set of challenge/response information that is used in various authentication protocols. For example, a username and password combination is the most familiar form of authentication credentials. Other forms of authentication credential may include various forms of challenge/response information, Public Key Infrastructure (PKI) certificates, smartcards, biometrics, etc. An authentication credential is differentiated from an authentication assertion: an authentication credential is presented by a user as part of an authentication protocol sequence with an authentication server or service, and an authentication assertion is a statement about the successful presentation and validation of a user's authentication credentials, subsequently transferred between entities when necessary.

Federation Model for Computing Environment that May Incorporate the Present Invention

In the context of the World Wide Web, users are coming to expect the ability to jump from interacting with an application on one Internet domain to another application on another domain with minimal regard to the information barriers between each particular domain. Users do not want the frustration that is caused by having to authenticate to multiple domains for a single transaction. In other words, users expect that organizations should interoperate, but users generally want domains to respect their privacy. In addition, users may prefer to limit the domains that permanently store private information. These user expectations exist in a rapidly evolving heterogeneous environment in which many enterprises and organizations are promulgating competing authentication techniques.

The present invention is supported within a federation model that allows enterprises to provide a single-sign-on experience to a user. In other words, the present invention may be implemented within a federated, heterogeneous environment. As an example of a transaction that would benefit from a federated, heterogeneous environment, referring again to FIG. **1**E, user **190** is able to authenticate to domain **191** and then have domain **191** provide the appropriate assertions to each downstream domain that might be involved in a transaction. These downstream domains need to be able to understand and trust authentication assertions and/or other types of assertions, even though there are no pre-established assertion formats between domain **191** and these other downstream domains. In addition to recognizing the assertions, the downstream domains need to be able to translate the identity contained within an assertion to an identity that represents user **190** within a particular domain, even though there is no pre-established identity mapping relationship. It should be noted, though, that the present invention is applicable to various types of domains and is not limited to ISP-type domains that are represented within FIG. **1**E as exemplary domains.

The present invention is supported within a federated environment. In general, an enterprise has its own user registry and maintains relationships with its own set of users. Each enterprise typically has its own means of authenticating these users. However, the federated scheme for use with the present invention allows enterprises to cooperate in a collective manner such that users in one enterprise can leverage relationships with a set of enterprises through an enterprise's participation in a federation of enterprises. Users can be granted access to resources at any of the federated enterprises as if they had a direct relationship with each enterprise. Users are not required to register at each business of interest, and users are not constantly required to identify and authenticate themselves. Hence, within this federated environment, an authentication scheme allows for a single-sign-on experience within the rapidly evolving heterogeneous environments in information technology.

In the context of the present invention, a federation is a set of distinct entities, such as enterprises, organizations, institutions, etc., that cooperate to provide a single-sign-on, ease-of-use experience to a user; a federated environment differs from a typical single-sign-on environment in that two enterprises need not have a direct, pre-established, relationship

defining how and what information to transfer about a user. Within a federated environment, entities provide services which deal with authenticating users, accepting authentication assertions, e.g., authentication tokens, that are presented by other entities, and providing some form of translation of the identity of the vouched-for user into one that is understood within the local entity.

Federation eases the administrative burden on service providers. A service provider can rely on its trust relationships with respect to the federation as a whole; the service provider does not need to manage authentication information, such as user password information, because it can rely on authentication that is accomplished by a user's authentication home domain or an identity provider.

The system that supports the present invention also concerns a federated identity management system that establishes a foundation in which loosely coupled authentication, user enrollment, user profile management and/or authorization services collaborate across security domains. Federated identity management allows services residing in disparate security domains to securely interoperate and collaborate even though there may be differences in the underlying security mechanisms and operating system platforms at these disparate domains.

Identity Provider vs. Service Provider

As mentioned above and as explained in more detail further below, a federated environment provides significant user benefits. A federated environment allows a user to authenticate at a first entity, which may act as an issuing party to issue an authentication assertion about the user for use at a second entity. The user can then access protected resources at a second, distinct entity, termed the relying party, by presenting the authentication assertion that was issued by the first entity without having to explicitly re-authenticate at the second entity. Information that is passed from an issuing party to a relying party is in the form of an assertion, and this assertion may contain different types of information in the form of statements. For example, an assertion may be a statement about the authenticated identity of a user, or it may be a statement about user attribute information that is associated with a particular user.

With reference now to FIG. **2**, a block diagram depicts the terminology of the federated environment with respect to a transaction that is initiated by a user to a first federated enterprise, which, in response, invokes actions at downstream entities within the federated environment. FIG. **2** shows that the terminology may differ depending on the perspective of an entity within the federation for a given federated operation. More specifically, FIG. **2** illustrates that a computing environment that supports the present invention supports the transitivity of trust and the transitivity of the authentication assertion process; a domain or an entity can issue an assertion based on its trust in an identity as asserted by another domain or another entity.

User **202** initiates a transaction through a request for a protected resource at enterprise **204**. If user **202** has been authenticated by enterprise **204** or will eventually be authenticated by enterprise **204** during the course of a transaction, then enterprise **204** may be termed the user's home domain for this federated session. Assuming that the transaction requires some type of operation by enterprise **206** and enterprise **204** transfers an assertion to enterprise **206**, then enterprise **204** is the issuing entity with respect to the particular operation, and enterprise **206** is the relying entity for the operation.

The issuing entity issues an assertion for use by the relying domain; an issuing entity is usually, but not necessarily, the user's home domain or the user's identity provider. Hence, it would usually be the case that the issuing party has authenticated the user using a typical authentication operation. However, it is possible that the issuing party has previously acted as a relying party whereby it received an assertion from a different issuing party. In other words, since a user-initiated transaction may cascade through a series of enterprises within a federated environment, a receiving party may subsequently act as an issuing party for a downstream transaction. In general, any entity that has the ability to issue authentication assertions on behalf of a user can act as an issuing entity.

The relying entity is an entity that receives an assertion from an issuing entity. The relying party is able to accept, trust, and understand an assertion that is issued by a third party on behalf of the user, i.e. the issuing entity; it is generally the relying entity's duty to use an appropriate authentication authority to interpret an authentication assertion. A relying party is an entity that relies on an assertion that is presented on behalf of a user or another entity. In this manner, a user can be given a single-sign-on experience at the relying entity instead of requiring the relying entity to prompt the user for the user's authentication credentials as part of an interactive session with the user.

Referring again to FIG. **2**, assuming that the transaction requires further operations such that enterprise **206** transfers an assertion to enterprise **208**, then enterprise **206** is an upstream entity that acts as the issuing entity with respect to the subsequent or secondary transaction operation, and enterprise **208** is a downstream entity that acts as the relying entity for the operation; in this case, enterprise **208** may be regarded as another downstream entity with respect to the original transaction, although the subsequent transaction can also be described with respect to only two entities.

As shown in FIG. **2**, a federated entity may act as a user's home domain, which provides identity information and attribute information about federated users. An entity within a federated computing environment that provides identity information, identity or authentication assertions, or identity services may be termed an identity provider. Other entities or federation partners within the same federation may rely on an identity provider for primary management of a user's authentication credentials, e.g., accepting a single-sign-on token that is provided by the user's identity provider; a domain at which the user authenticates may be termed the user's (authentication) home domain. The identity provider may be physically supported by the user's employer, the user's ISP, or some other commercial entity.

An identity provider is a specific type of service that provides identity information as a service to other entities within a federated computing environment. With respect to most federated transactions, an issuing party for an authentication assertion would usually be an identity provider; any other entity can be distinguished from the identity provider. Any other entity that provides a service within the federated computing environment can be categorized as a service provider. Once a user has authenticated to the identity provider, other entities or enterprises in the federation may be regarded as merely service providers for the duration of a given federated session or a given federated transaction.

In some circumstances, there may be multiple entities within a federated environment that may act as identity providers for a user. For example, the user may have accounts at multiple federated domains, each of which is able to act as an identity provider for the user; these domains do not necessarily have information about the other domains nor about a user's identity at a different domain.

Although it may be possible that there could be multiple enterprises within a federated environment that may act as identity providers, e.g., because there may be multiple enterprises that have the ability to generate and validate a user's authentication credentials, etc., a federated transaction usually involves only a single identity provider. If there is only a single federated entity that is able to authenticate a user, e.g., because there is one and only one entity within the federation with which the user has performed a federated enrollment or registration operation, then it would be expected that this entity would act as the user's identity provider in order to support the user's transactions throughout the federated environment.

Within some federated transactions that require the interoperation of multiple service providers, a downstream service provider may accept an assertion from an upstream service provider; the conditions in which an upstream service provider may act as an issuing entity to a downstream service provider that is acting as a relying party may depend upon the type of trust relationship between the service providers and the type of transaction between the service providers. Within the scope of a simple federated transaction, however, there is only one entity that acts as an issuing entity.

The present invention may be supported within a given computing environment in which a federated infrastructure can be added to existing systems while minimizing the impact on an existing, non-federated architecture. Hence, operations, including authentication operations, at any given enterprise or service provider are not necessarily altered by the fact that an entity may also participate within a federated environment. In other words, even though an entity's computing systems may be integrated into a federated environment, a user may be able to continue to perform various operations, including authentication operations, directly with an enterprise in a non-federated manner. However, the user may be able to have the same end-user experience while performing a federated operation with respect to a given entity as if the user had performed a similar operation with the given entity in a non-federated manner. Hence, it should be noted that not all of a given enterprise's users necessarily participate federated transactions when the given enterprise participates in a federation; some of the enterprise's users may interact with the enterprise's computing systems without performing any federated transactions.

Moreover, user registration within the computing environment of a given enterprise, e.g., establishment of a user account in a computer system, is not necessarily altered by the fact that the enterprise may also participate within a federated environment. For example, a user may still establish an account at a domain through a legacy or pre-existing registration process that is independent of a federated environment. Hence, in some cases, the establishment of a user account at an enterprise may or may not include the establishment of account information that is valid across a federation when the enterprise participates within a federated computing environment.

Federated Architecture—Federation Front-End for Legacy Systems

With reference now to FIG. **3**, a block diagram depicts the integration of pre-existing data processing systems at a given domain with some federated architecture components that may be used to support an embodiment of the present invention. A federated environment includes federated entities that provide a variety of services for users. User **312** interacts with client device **314**, which may support browser application **216** and various other client applications **318**. User **312** is distinct from client device **314**, browser **316**, or any other software that acts as interface between user and other devices and services. In some cases, the following description may make a distinction between the user acting explicitly within a client application and a client application that is acting on behalf of the user. In general, though, a requester is an intermediary, such as a client-based application, browser, SOAP client, etc., that may be assumed to act on behalf of the user.

Browser application **316** may be a typical browser, including those found on mobile devices, that comprises many modules, such as HTTP communication component **320** and markup language (ML) interpreter **322**. Browser application **316** may also support plug-ins, such as web services client **324**, and/or downloadable applets, which may or may not require a virtual machine runtime environment. Web services client **324** may use Simple Object Access Protocol (SOAP), which is a lightweight protocol for defining the exchange of structured and typed information in a decentralized, distributed environment. SOAP is an XML-based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it; a set of encoding rules for expressing instances of application-defined datatypes; and a convention for representing remote procedure calls and responses. User **312** may access web-based services using browser application **316**, but user **312** may also access web services through other web service clients on client device **314**. Some of the federated operations may employ HTTP redirection via the user's browser to exchange information between entities in a federated environment. However, it should be noted that the present invention may be supported over a variety of communication protocols and is not meant to be limited to HTTP-based communications. For example, the entities in the federated environment may communicate directly when necessary; messages are not required to be redirected through the user's browser.

The present invention may be supported in a manner such that components that are required for a federated environment can be integrated with pre-existing systems. FIG. **3** depicts one embodiment for implementing these components as a front-end to a pre-existing system. The pre-existing components at a federated domain can be considered as legacy applications or back-end processing components **330**, which include authentication service runtime (ASR) servers **332** in a manner similar to that shown in FIG. **4**. ASR servers **332** are responsible for authenticating users when the domain controls access to application servers **334**, which can be considered to generate, retrieve, or otherwise support or process protected resources **335**. The domain may continue to use legacy user registration application **336** to register users for access to application servers **334**. Information that is needed to authenticate a registered user with respect to legacy operations is stored in enterprise user registry **338**; enterprise user registry **338** may be accessible to federation components as well.

After joining a federated environment, the domain may continue to operate without the intervention of federated components. In other words, the domain may be configured so that users may continue to access particular application servers or other protected resources directly without going through a point-of-contact server or other component implementing this point-of-contact server functionality; a user that accesses a system in this manner would experience typical authentication flows and typical access. In doing so, however, a user that directly accesses the legacy system would not be able to establish a federated session that is known to the domain's point-of-contact server.

The domain's legacy functionality can be integrated into a federated environment through the use of federation front-end processing **340**, which includes point-of-contact server **342** and trust proxy server **344** (or more simply, trust proxy **344** or trust service **344**) which itself interacts with Security Token Service (STS) **346**, which are described in more detail below with respect to FIG. **4**. Federation configuration application **348** allows an administrative user to configure the federation front-end components to allow them to interface with the legacy back-end components through federation interface unit **350**. Federated functionality may be implemented in distinct system components or modules. In a preferred embodiment, most of the functionality for performing federation operations may be implemented by a collection of logical components within a single federation application; federated user lifecycle management application **352** includes trust service **344** along with single-sign-on protocol service (SPS) **354**. Trust service **344** may comprise identity-and-attribute service (I&AS) **356**, which is responsible for identity mapping operations, attribute retrieval, etc., as part of federation functionality. Identity-and-attribute service **356** may also be employed by single-sign-on protocol service **354** during single-sign-on operations. A federation user registry **358** may be employed in certain circumstances to maintain user-related information for federation-specific purposes.

Legacy or pre-existing authentication services at a given enterprise may use various, well known, authentication methods or tokens, such as username/password or smart card token-based information. However, in a preferred federated computing system for supporting the present invention, the functionality of a legacy authentication service can be used in a federated environment through the use of point-of-contact servers. Users may continue to access a legacy authentication server directly without going through a point-of-contact server, although a user that accesses a system in this manner would experience typical authentication flows and typical access; a user that directly accesses a legacy authentication system would not be able to generate a federated authentication assertion as proof of identity in accordance with the present invention. One of the roles of the federation front-end is to translate a federated authentication token received at a point-of-contact server into a format understood by a legacy authentication service. Hence, a user accessing the federated environment via the point-of-contact server would not necessarily be required to re-authenticate to the legacy authentication service. Preferably, the user would be authenticated to a legacy authentication service by a combination of the point-of-contact server and a trust proxy such that it appears as if the user was engaged in an authentication dialog.

Federated Architecture—Point-of-Contact Servers, Trust Proxies, and Trust Brokers

With reference now to FIG. **4**, a block diagram depicts an example of a manner in which some components within a federated architecture may be used to establish trust relationships to support an implementation of the present invention. A federated environment includes federated enterprises or similar entities that provide a variety of services for users. A user, through an application on a client device, may attempt to access resources at various entities, such as enterprise **410**. A point-of-contact server at each federated enterprise, such as point-of-contact (POC) server **412** at enterprise **410**, is the entry point into the federated environment for requests from a client to access resources that are supported and made available by enterprise **410**. The point-of-contact server minimizes the impact on existing components within an existing, non-federated architecture, e.g., legacy systems, because the point-of-contact server handles many of the federation requirements. The point-of-contact server provides session management, protocol conversion, and possibly initiates authentication and/or attribute assertion conversion. For example, the point-of-contact server may translate HTTP or HTTPS messages to SOAP and vice versa. As explained in more detail further below, the point-of-contact server may also be used to invoke a trust proxy to translate assertions, e.g., a SAML token received from an issuing party can be translated into a Kerberos token understood by a receiving party.

A trust service (also termed a trust proxy, a trust proxy server, or a trust service), such as trust proxy (TP) **414** at enterprise **410**, establishes and maintains a trust relationship between two entities in a federation. A trust service generally has the ability to handle authentication token format translation (through the security token service, which is described in more detail further below) from a format used by the issuing party to one understood by the receiving party.

Together, the use of a point-of-contact server and a trust service minimize the impact of implementing a federated architecture on an existing, non-federated set of systems. Hence, the exemplary federated architecture requires the implementation of at least one point-of-contact server and at least one trust service per federated entity, whether the entity is an enterprise, a domain, or other logical or physical entity. The exemplary federated architecture, though, does not necessarily require any changes to the existing, non-federated set of systems. Preferably, there is a single trust service for a given federated entity, although there may be multiple instances of a trust service component for availability purposes, or there may be multiple trust services for a variety of smaller entities within a federated entity, e.g., separate subsidiaries within an enterprise. It is possible that a given entity could belong to more than one federation, although this scenario would not necessarily require multiple trust services as a single trust service may be able to manage trust relationships within multiple federations.

One role of a trust service may be to determine or to be responsible for determining the required token type by another domain and/or the trust service in that domain. A trust service has the ability or the responsibility to handle authentication token format translation from a format used by the issuing party to one understood by the receiving party. Trust service **414** may also be responsible for any user identity translation or attribute translation that occurs for enterprise **410**, or this responsibility may be supported by a distinct identity-and-attribute service, e.g., such as identity-and-attribute service **356** as shown in FIG. **3**. In addition, a trust service can support the implementation of aliases as representatives of a user identity that uniquely identify a user without providing any addition information about the user's real world identity. Furthermore, a trust proxy can issue authorization and/or session credentials for use by the point-of-contact server. However, a trust service may invoke a trust broker for assistance, as described further below. Identity translation may be required to map a user's identity and attributes as known to an issuing party to one that is meaningful to a receiving party. This translation may be invoked by either a trust service at an issuing entity, a trust service at a receiving entity, or both.

Trust service **414**, or a distinct identity-and-attribute service as mentioned above, may include (or interact with) an internalized component, shown as security token service (STS) component **416**, which will provide token translation and will invoke authentication service runtime (ASR) **418** to validate and generate tokens. The security token service provides the token issuance and validation services required by

the trust service, which may include identity translation. The security token service therefore includes an interface to existing authentication service runtimes, or it incorporates authentication service runtimes into the service itself. Rather than being internalized within the trust service, the security token service component may also be implemented as a stand-alone component, e.g., to be invoked by the trust service, or it may be internalized within a transaction server, e.g., as part of an application server.

For example, an security token service component may receive a request to issue a Kerberos token. As part of the authentication information of the user for whom the token is to be created, the request may contain a binary token containing a username and password. The security token service component will validate the username and password against, e.g., an LDAP runtime (typical authentication) and will invoke a Kerberos KDC (Key Distribution Center) to generate a Kerberos ticket for this user. This token is returned to the trust service for use within the enterprise; however, this use may include externalizing the token for transfer to another domain in the federation.

In a manner similar to that described with respect to FIG. **1**D, a user may desire to access resources at multiple enterprises within a federated environment, such as both enterprise **410** and enterprise **420**. In a manner similar to that described above for enterprise **410**, enterprise **420** comprises point-of-contact server **422**, trust service **424**, security token service (STS) **426**, and authentication service runtime **428**. Although the user may directly initiate separate transactions with each enterprise, the user may initiate a transaction with enterprise **410** which cascades throughout the federated environment. Enterprise **410** may require collaboration with multiple other enterprises within the federated environment, such as enterprise **420**, to complete a particular transaction, even though the user may not have been aware of this necessity when the user initiated a transaction. Enterprise **420** becomes involved as a downstream entity, and enterprise **410** may present a assertion to enterprise **420** if necessary in order to further the user's federated transaction.

It may be the case that a trust service does not know how to interpret the authentication token that is received by an associated point-of-contact server and/or how to translate a given user identity and attributes. In this case, the trust service may choose to invoke functionality at a trust broker component, such as trust broker **430**. A trust broker maintains relationships with individual trust proxies/services, thereby providing transitive trust between trust services. Using a trust broker allows each entity within a federated environment, such enterprises **410** and **420**, to establish a trust relationship with the trust broker rather than establishing multiple individual trust relationships with each entity in the federated environment. For example, when enterprise **420** becomes involved as a downstream entity for a transaction initiated by a user at enterprise **410**, trust service **414** at enterprise **410** can be assured that trust service **424** at enterprise **420** can understand an assertion from trust service **414** by invoking assistance at trust broker **430** if necessary. Although FIG. **4** depicts the federated environment with a single trust broker, a federated environment may have multiple trust brokers.

It should be noted that although FIG. **4** depicts point-of-contact server **412**, trust service **414**, security token service component **416**, and authentication service runtime **418** as distinct entities, it is not necessary for these components to be implemented on separate components. For example, it is possible for the functionality of these separate components to be implemented as a single application, as applications on a single physical device, or as distributed applications on multiple physical devices. In addition, FIG. **4** depicts a single point-of-contact server, a single trust service, and a single security token server for an enterprise, but an alternative configuration may include multiple point-of-contact servers, multiple trust services, and multiple security token servers for each enterprise. The point-of-contact server, the trust service, the security token service, and other federated entities may be implemented in various forms, such as software applications, objects, modules, software libraries, etc.

A trust service/STS may be capable of accepting and validating many different authentication credentials, including traditional credentials such as a username and password combinations and Kerberos tickets, and federated authentication token formats, including authentication tokens produced by a third party. A trust service/STS may allow the acceptance of an authentication token as proof of authentication elsewhere. The authentication token is produced by an issuing party and is used to indicate that a user has already authenticated to that issuing party. The issuing party produces the authentication token as a means of asserting the authenticated identity of a user. A trust service/STS is also able to process attribute tokens or tokens that are used to secure communication sessions or conversations, e.g., those that are used to manage session information in a manner similar to an SSL session identifier.

A security token service invokes an authentication service runtime as necessary. The authentication service runtime supports an authentication service capable of authenticating a user. The authentication service acts as an authentication authority that provides indications of successful or failed authentication attempts via authentication responses. The trust service/STS may internalize an authentication service, e.g., a scenario in which there is a brand-new installation of a web service that does not need to interact with an existing legacy infrastructure. Otherwise, the security token service component will invoke external authentication services for validation of authentication tokens. For example, the security token service component could "unpack" a binary token containing a username/password and then use an LDAP service to access a user registry to validate the presented credentials.

When used by another component such as an application server, the security token service component can be used to produce tokens required for single-sign-on to legacy authentication systems; this functionality may be combined with or replaced by functionality within a single-sign-on protocol service, such as SPS **354** that is shown in FIG. **3**. Hence, the security token service component can be used for token translation for internal purposes, i.e. within an enterprise, and for external purposes, i.e. across enterprises in a federation. As an example of an internal purpose, a Web application server may interface to a mainframe via an IBM CICS (Customer Information Control System) transaction gateway; CICS is a family of application servers and connectors that provides enterprise-level online transaction management and connectivity for mission-critical applications. The Web application server may invoke the security token service component to translate a Kerberos ticket (as used internally by the Web application server) to an IBM RACF® passticket required by the CICS transaction gateway.

The entities that are shown in FIG. **4** can be explained using the terminology that was introduced above, e.g., "identity provider" and "service provider". As part of establishing and maintaining trust relationships, an identity provider's trust service can determine what token types are required/accepted by a service provider's trust service. Thus, trust services use this information when invoking token services from a security token service. When an identity provider's trust service is

required to produce an authentication assertion for a service provider, the trust service determines the required token type and requests the appropriate token from the security token service.

When a service provider's trust service receives an authentication assertion from an identity provider, the trust service knows what type of assertion that it expected and what type of assertion that it needs for internal use within the service provider. The service provider's trust service therefore requests that the security token service generate the required internal-use token based on the token in the received authentication assertion.

Both trust services and trust brokers have the ability to translate an assertion received from an identity provider into a format that is understood by a service provider. The trust broker has the ability to interpret the assertion format (or formats) for each of the trust services with whom there is a direct trust relationship, thereby allowing the trust broker to provide assertion translation between an identity provider and a service provider. This translation can be requested by either party through its local trust service. Thus, the identity provider's trust service can request translation of an assertion before it is sent to the service provider. Likewise, the service provider's trust service can request translation of an assertion received from an identity provider.

Assertion translation comprises user identity translation, authentication assertion translation, attribute assertion translation, or other forms of assertion translation. Reiterating the point above, assertion translation is handled by the trust components within a federation, e.g., trust services and trust brokers. A trust service may perform the translation locally, either at the identity provider or at the service provider, or a trust service may invoke assistance from a trust broker.

Assuming that an identity provider and a service provider already have individual trust relationships with a trust broker, the trust broker can dynamically create, i.e. broker, new trust relationships between issuing parties and relying parties if necessary. After the initial trust relationship brokering operation that is provided by the trust broker, the identity provider and the service provider may directly maintain the relationship so that the trust broker need not be invoked for future translation requirements. It should be noted that translation of authentication tokens can happen at three possible places: the identity provider's trust service, the service provider's trust service, and the trust broker. Preferably, the identity provider's trust service generates an authentication assertion that is understood by the trust broker to send to the service provider. The service provider then requests a translation of this token from the trust broker into a format recognizable by the service provider. Token translation may occur before transmission, after transmission, or both before and after transmission of the authentication assertion.

Trust Relationships within Federated Architecture

Within an exemplary federated environment that is able to support the present invention, there are two types of "trust domains" that must be managed: enterprise trust domains and federation trust domains. The differences between these two types of trust domain are based in part on the business agreements governing the trust relationships with the trust domain and the technology used to establish trust. An enterprise trust domain contains those components that are managed by the enterprise; all components within that trust domain may implicitly trust each other. In general, there are no business agreements required to establish trust within an enterprise because the deployed technology creates inherent trust within an enterprise, e.g., by requiring mutually authenticated SSL sessions between components or by placing components within a single, tightly controlled data center such that physical control and proximity demonstrate implicit trust. Referring to FIG. 2B, the legacy applications and back-end processing systems may represent an enterprise trust domain, wherein the components communicate on a secure internal network.

Federation trust domains are those that cross enterprise boundaries; from one perspective, a federation trust domain may represent trust relationships between distinct enterprise trust domains. Federation trust domains are established through trust proxies across enterprise boundaries between federation partners. Trust relationships involve some sort of a bootstrapping process by which initial trust is established between trust proxies. Part of this bootstrap process may include the establishment of shared secret keys and rules that define the expected and/or allowed token types and identifier translations. In general, this bootstrapping process can be implemented out-of-band as this process may also include the establishment of business agreements that govern an enterprise's participation in a federation and the liabilities associated with this participation.

There are a number of possible mechanisms for establishing trust in a federated business model. In a federation model, a fundamental notion of trust between the federation participants is required for business reasons in order to provide a level of assurance that the assertions (including tokens and attribute information) that are transferred between the participants are valid. If there is no trust relationship, then the service provider cannot depend upon the assertions received from the identity provider; they cannot be used by the service provider to determine how to interpret any information received from the identity provider.

For example, a large corporation may want to link several thousand global customers, and the corporation could use non-federated solutions. As a first example, the corporation could require global customers to use a digital certificate from a commercial certificate authority to establish mutual trust. The commercial certificate authority enables the servers at the corporation to trust servers located at each of the global customers. As a second example, the corporation could implement third-party trust using Kerberos; the corporation and its global customers could implement a trusted third-party Kerberos domain service that implements shared-secret-based trust. As a third example, the corporation could establish a private scheme with a proprietary security message token that is mutually trusted by the servers of its global customers.

Any one of these approaches may be acceptable if the corporation needed to manage trust relationships with a small number of global customers, but this may become unmanageable if there are hundreds or thousands of potential federation partners. For example, while it may be possible for the corporation to force its smaller partners to implement a private scheme, it is unlikely that the corporation will be able to impose many requirements on its larger partners.

An enterprise may employ trust relationships established and maintained through trust proxies and possibly trust brokers. An advantage of the exemplary federated architecture that is shown in the figures is that it does not impose additional requirements above and beyond the current infrastructures of an enterprise and its potential federation partners.

However, this exemplary federation architecture does not relieve an enterprise and its potential federation partners from the preliminary work required to establish business and liability agreements that are required for participation in the federation. In addition, the participants cannot ignore the technological bootstrapping of a trust relationship. The

exemplary federation architecture allows this bootstrapping to be flexible, e.g., a first federation partner can issue a Kerberos ticket with certain information, while a second federation partner can issue a SAML authentication assertion with certain information.

In the exemplary federation architecture, the trust relationships are managed by the trust proxies, which may include (or may interact with) a security token service that validates and translates a token that is received from an identity provider based on the pre-established relationship between two trust proxies. In situations where it is not feasible for a federated enterprise to establish trust relationships (and token translation) with another federated enterprise, a trust broker may be invoked; however, the federated enterprise would need to establish a relationship with a trust broker.

With reference now to FIG. **5**, a block diagram depicts an exemplary set of trust relationships between federated domains using trust proxies and a trust broker in accordance with an exemplary federated architecture that is able to support the present invention. Although FIG. **4** introduced the trust broker, FIG. **5** illustrates the importance of transitive trust relationships within the exemplary federated architecture.

Federated domains **502-506** incorporate trust proxies **508-512**, respectively. Trust proxy **508** has direct trust relationship **514** with trust proxy **510**. Trust broker **520** has direct trust relationship **516** with trust proxy **510**, and trust broker **520** has direct trust relationship **518** with trust proxy **512**. Trust broker **520** is used to establish, on behalf of a federation participant, a trust relationship based on transitive trust with other federation partners. The principle of transitive trust allows trust proxy **510** and trust proxy **512** to have brokered trust relationship **522** via trust broker **520**. Neither trust proxy **510** nor **512** need to know how to translate or validate the other's assertions; the trust broker may be invoked to translate an assertion into one that is valid, trusted, and understood at the other trust proxy.

Business agreements that specify contractual obligations and liabilities with respect to the trust relationships between federated enterprises can be expressed in XML through the use of the ebXML (Electronic Business using XML) standards. For example, a direct trust relationship could be represented in an ebXML document; each federated domain that shares a direct trust relationship would have a copy of a contract that is expressed as an ebXML document. Operational characteristics for various entities within a federation may be specified within ebXML choreographies and published within ebXML registries; any enterprise that wishes to participate in a particular federation, e.g., to operate a trust proxy or trust broker, would need to conform to the published requirements that were specified by that particular federation for all trust proxies or trust brokers within the federation. A security token service could parse these ebXML documents for operational details on the manner in which tokens from other domains are to be translated. It should be noted, though, that other standards and mechanisms could be employed to support the present invention for specifying the details about the manner in which the trust relationships within a federation are implemented.

Single-Sign-on within Federated Architecture

During a given user's session, the user may visit many federated domains to use the web services that are offered by those domains. Domains can publish descriptions of services that they provide using standard specifications such as UDDI and WSDL, both of which use XML as a common data format. The user finds the available services and service providers through applications that also adhere to these standard specifications. SOAP provides a paradigm for communicating requests and responses that are expressed in XML. Entities within a federated environment may employ these standards among others.

Within a federation, a user expects to have a single-sign-on experience in which the user completes a single authentication operation, and this authentication operation suffices for the duration of the user's session, regardless of the federation partners visited during that session. A session can be defined as the set of transactions from (and including) the initial user authentication, i.e. logon, to logout. Within a session, a user's actions will be governed in part by the privileges granted to the user for that session.

The federated architecture that is described hereinabove supports single-sign-on operations. To facilitate a single-sign-on experience, web services that support the federated environment will also support using an authentication assertion or security token generated by a third-party to provide proof of authentication of a user. This assertion will contain some sort of evidence of the user's successful authentication to the issuing party together with an identifier for that user. For example, a user may complete traditional authentication operation with one federation partner, e.g., by providing a username and password that the federation partners uses to build authentication credentials for the user, and then the federation partner is able to provide a SAML authentication assertion that is generated by the authenticating/issuing party to a different federation partner.

The federated environment also allows web services or other applications to request web services, and these web services would also be authenticated. Authentication in a web services environment is the act of verifying the claimed identity of the web services request so that the enterprise can restrict access to authorized clients. A user who requests or invokes a web service would almost always authenticated, so the need for authentication within a federated environment that supports the present invention is not any different from current requirements of web services for user authentication.

Authentication of users that are accessing the computational resources of an enterprise without participating in a federated session are not impacted by the presence of a federated infrastructure. For example, an existing user who authenticates with a forms-based authentication mechanism over HTTP/S to access non-federated resources at a particular domain is not affected by the introduction of support at the domain for the federated environment. Authentication is handled in part by a point-of-contact server, which in turn may invoke a separate trust proxy or trust service component; the use of a point-of-contact server minimizes the impact on the infrastructure of an existing domain. For example, the point-of-contact server can be configured to pass through all non-federated requests to be handled by the back-end or legacy applications and systems at the domain.

The point-of-contact server may choose to invoke an HTTP-based authentication method, such as basic authentication, forms-based authentication, or some other authentication method. The point-of-contact server also supports a federation domain by recognizing an assertion that has been presented by the user as proof of authentication, such as an SAML authentication assertion, wherein the assertion has crossed between enterprise domains. The point-of-contact server may invoke the trust service, which in turn may invoke its security token service for validation of authentication credentials/security tokens.

Authentication of web services or other applications comprises the same process as authentication of users. Requests from web services carry a security token containing an

authentication assertion, and this security token would be validated by the trust service in the same manner as a token presented by a user. A request from a web service should be accompanied by this token because the web service would have discovered what authentication assertions/security tokens were required by the requested service as advertised in UDDI.

With reference now to FIG. **6**, a block diagram depicts a federated environment that supports federated single-sign-on operations. User **600**, through a client device and an appropriate client application, such as a browser, desires to access a web service that is provided by enterprise/domain **610**, which supports data processing systems that act as a federated domain within a federated environment. Domain **610** supports point-of-contact server **612** and trust proxy or trust service **614**; similarly, domain **620** supports point-of-contact server **622** and trust proxy or trust service **624**, while domain **630** supports point-of-contact server **632** and trust proxy or trust service **634**. The trust proxies/services rely upon trust broker **650** for assistance, as described above. Additional domains and trust proxies/services may participate in the federated environment. FIG. **6** is used to describe a federated single-sign-on operation between domain **610** and domain **620**; a similar operation may occur between domain **610** and domain **630**.

The user completes an authentication operation with respect to domain **610**; this authentication operation is handled by point-of-contact server **612**. The authentication operation is triggered when the user requests access to some resource that requires an authenticated identity, e.g., for access control purposes or for personalization purposes. Point-of-contact server **612** may invoke a legacy authentication service, or it may invoke trust proxy **614** to validate the user's presented authentication credentials. Domain **610** becomes the user's identity provider or home domain for the duration of the user's federated session.

At some later point in time, the user initiates a transaction at a federation partner, such as enterprise **620** that also supports a federated domain, thereby triggering a federated single-sign-on operation. For example, a user may initiate a new transaction at domain **620**, or the user's original transaction may cascade into one or more additional transactions at other domains. As another example, the user may invoke a federated single-sign-on operation to a resource in domain **620** via point-of-contact server **612**, e.g., by selecting a special link on a web page that is hosted within domain **610** or by requesting a portal page that is hosted within domain **610** but that displays resources hosted in domain **620**. Point-of-contact server **612** sends a request to trust proxy **614** to generated a federation single-sign-on token for the user that is formatted to be understood or trusted by domain **620**. Trust proxy **614** returns this token to point-of-contact server **612**, which sends this token to point-of-contact server **622** in domain. Domain **610** acts as an issuing party for the user at domain **620**, which acts as a relying party. The user's token would be transferred with the user's request to domain **620**; this token may be sent using HTTP redirection via the user's browser, or it may be sent by invoking the request directly of point-of-contact server **622** (over HTTP or SOAP-over-HTTP) on behalf of the user identified in the token supplied by trust proxy **614**.

Point-of-contact server **622** receives the request together with the federation single-sign-on token and invokes trust proxy **624**. Trust proxy **624** receives the federation single-sign-on token, validates the token, and assuming that the token is valid and trusted, generates a locally valid token for the user. Trust proxy **624** returns the locally valid token to point-of-contact server **622**, which establishes a session for the user within domain **620**. If necessary, point-of-contact server **622** can initiate a federated single-sign-on at another federated partner.

Validation of the token at domain **620** is handled by the trust proxy **624**, possibly with assistance from a security token service. Depending on the type of token presented by domain **610**, the security token service may need to access a user registry at domain **620**. For example, domain **620** may provide a binary security token containing the user's name and password to be validated against the user registry at domain **620**. Hence, in this example, an enterprise simply validates the security token from a federated partner. The trust relationship between domains **610** and **620** ensures that domain **620** can understand and trust the security token presented by domain **610** on behalf of the user.

Federated single-sign-on requires not only the validation of the security token that is presented to a relying domain on behalf of the user but the determination of a locally valid user identifier at the relying domain based on information contained in the security token. One result of a direct trust relationship and the business agreements required to establish such a relationship is that at least one party, either the issuing domain or the relying domain or both, will know how to translate the information provided by the issuing domain into an identifier valid at the relying domain. In the brief example above, it was assumed that the issuing domain, i.e. domain **610**, is able to provide the relying domain, i.e. domain **620**, with a user identifier that is valid in domain **620**. In that scenario, the relying domain did not need to invoke any identity mapping functionality. Trust proxy **624** at domain **620** will generate a security token for the user that will "vouch-for" this user. The types of tokens that are accepted, the signatures that are required on tokens, and other requirements are all pre-established as part of the federation's business agreements. The rules and algorithms that govern identifier translation are also pre-established as part of the federation's business agreements. In the case of a direct trust relationship between two participants, the identifier translation algorithms will have been established for those two parties and may not be relevant for any other parties in the federation.

However, it is not always the case that the issuing domain will know how to map the user from a local identifier for domain **610** to a local identifier for domain **620**. In some cases, it may be the relying domain that knows how to do this mapping, while in yet other cases, neither party will know how to do this translation, in which case a third party trust broker may need to be invoked. In other words, in the case of a brokered trust relationship, the issuing and relying domains do not have a direct trust relationship with each other. They will, however, have a direct trust relationship with a trust broker, such as trust broker **650**. Identifier mapping rules and algorithms will have been established as part of this relationship, and the trust broker will use this information to assist in the identifier translation that is required for a brokered trust relationship.

Domain **620** receives the token that is issued by domain **610** at point-of-contract server **622**, which invokes trust proxy **624** to validate the token and perform identity mapping. In this case, since trust proxy **624** is not able to map the user from a local identifier for domain **610** to a local identifier for domain **620**, trust proxy **624** invokes trust broker **650**, which validates the token and performs the identifier mapping. After obtaining the local identifier for the user, trust proxy **624**, possibly through its security token service, can generate any local tokens that are required by the back-end applications at domain **620**, e.g., a Kerberos token may be required to facilitate single-sign-on from the point-of-contact server to the

application server. After obtaining a locally valid token, if required, the point-of-contact server is able to build a local session for the user. The point-of-contract server will also handle coarse-grained authorization of user requests and forward the authorized requests to the appropriate application servers within domain **620**.

Federated User Lifecycle Management

A portion of the above description of FIGS. **2-6** explained an organization of components that may be used in a federated environment while other portions explained the processes for supporting single-sign-on operations across the federated environment. Service providers or relying domains within a federated environment do not necessarily have to manage a user's authentication credentials, and those relying domains can leverage a single single-sign-on token that is provided by the user's identity provider or home domain. The description of FIGS. **2-6** above, though, does not explain an explicit process by which federated user lifecycle management may be accomplished in an advantageous manner at the federated domains of federation partners.

Federated user lifecycle management functionality/service comprises functions for supporting or managing federated operations with respect to the particular user accounts or user profiles of a given user at multiple federated domains; in some cases, the functions or operations are limited to a given federated session for the user. In other words, federated user lifecycle management functionality refers to the functions that allow management of federated operations across a plurality of federated partners, possibly only during the lifecycle of a single user session within a federated computing environment.

Each federated domain might manage a user account, a user profile, or a user session of some kind with respect to the functions at each respective federated domain. For example, a particular federated domain might not manage a local user account or user profile within the particular federated domain, but the federated domain might manage a local user session for a federated transaction after the successful completion of a single-sign-on operation at the federated domain. As part of the federated user lifecycle management functionality that is supported by that particular federated domain, the federated domain can participate in a single-sign-off operation that allows the federated domain to terminate the local user session after the federated transaction is complete, thereby improving security and promoting efficient use of resources.

In another example of the use of federated user lifecycle management functionality, a user may engage in an online transaction that requires the participation of multiple federated domains. A federated domain might locally manage a user profile in order to tailor the user's experience with respect to the federated domain during each of the user's federated sessions that involve the federated domain. As part of the federated user lifecycle management functionality that is supported by that particular federated domain, the information in the federated domain's local user profile can be used in a seamless manner during a given federated transaction with information from other profiles at other federated domains that are participating in the given federated transaction. For example, the information from the user's multiple local user profiles might be combined in some type of merging operation such that the user's information is visually presented to the user, e.g., within a web page, in a manner such that the user is not aware of the different origins or sources of the user's information.

Federated user lifecycle management functionality may also comprise functions for account linking/delinking. A user is provided with a common unique user identifier across federation partners, which enables single-sign-on and the retrieval of attributes (if necessary) about a user as part of the fulfillment of a request at one federation partner. Furthermore, the federation partner can request additional attributes from an identity provider using the common unique user identifier to refer to the user in an anonymous manner.

With reference now to FIG. **7**, a block diagram depicts some of the components in a federated domain for implementing federated user lifecycle management functionality in order to support the present invention. FIG. **7** depicts elements at a single federated domain, such as the federated domain that is shown in FIG. **3**. Some of the elements in FIG. **7** are similar or identical to elements that have been discussed hereinabove with respect to other figures, such as FIG. **3**: point-of-contact server/service **702** is equivalent to point-of-contact server **342**; application servers **704**, which run services that control access to protected resources, are equivalent to application servers **334**; protected or controlled resources **706** are equivalent to protected resources **335**; and federated user lifecycle management (FULM) application **708** is equivalent to federated user lifecycle management application **352**. Although firewalls were not illustrated within FIG. **3**, firewalls are illustrated within FIG. **7**. Firewall **710** and firewall **712** create an external DMZ (electronic DeMilitarized Zone) that protects the enterprise's computing environment from computing threats outside of the enterprise's domain, e.g., via the Internet.

The different perspectives that are shown in FIG. **3** and FIG. **7** are not incompatible or at cross-purposes. In contrast with the example that is shown in FIG. **7**, FIG. **3** does not illustrate the firewalls, yet point-of-contact server **342** resides within federation front-end **340**; in addition, federated user lifecycle management application **352** is contained within federation front-end **340**. In FIG. **7**, point-of-contact server **702** is illustrated as residing within the DMZ between firewalls **710** and **712**, which form an electronic or physical front-end to the enterprise domain; in addition, federated user lifecycle management application/service **708** resides electronically behind firewall **712**. Trust service **714**, single-sign-on protocol service **716**, and identity-and-attribute service **718** employ enterprise user registry **720** and federation user registry **722** as necessary. The different perspectives of FIG. **3** and FIG. **7** can be reconciled by regarding federation front-end **340** and back-end **330** in FIG. **3** as a logical organization of components while regarding the DMZ and the other components in FIG. **7** as forming a physical or electronic front-end and a physical or electronic back-end, either of which may contain federated components.

Reiterating the roles of a point-of-contact entity/service, the point-of-contact entity provides session management, at least with respect to a user's interaction with the federation functionality with an enterprise's computing environment; applications within a legacy back-end of the enterprise's computing environment may also implement its own session management functionality. Assuming that an enterprise implements policy functionality with respect to the federated computing environment, the point-of-contact entity may act as a policy enforcement point to some other federation partner's policy decision point. In addition, assuming that it is permissible given the implementation of the federation functionality, the point-of-contact entity is responsible for initiating a direction authentication operation against a user in those scenarios in which a single-sign-on operation is not employed. As such, the point-of-contact entity may be implemented in a variety of forms, e.g., as a reverse proxy server, as a web server plug-in, or in some other manner. The point-of-contact functionality may also be implemented within an

application server itself, in which case the federated user lifecycle management services may be logically located within the DMZ.

More importantly, referring again to FIG. **7**, federated user lifecycle management application **708** also comprises support for interfacing to, interacting with, or otherwise interoperating with federated user lifecycle management plug-ins **724**, which are not shown in FIG. **3**. In the exemplary architecture that is shown in FIG. **7**, federated protocol runtime plug-ins provide the functionality for various types of independently published or developed federated user lifecycle management standards or profiles, such as: WS-Federation Passive Client; and Liberty Alliance ID-FF Single Sign On (B/A, B/P and LECP), Register Name Identifier, Federation Termination Notification, and Single Logout. Different sets of federated protocols may be accessed at different URI's. This approach allows the federated user lifecycle management application to concurrently support multiple standards or specifications of federated user lifecycle management, e.g., the WS-Federation web services specification versus the Liberty Alliance's specifications, within a single application, thereby minimizing the configuration impact on the overall environment for supporting different federation protocols.

More specifically, the appropriate federated user lifecycle management functionality is invoked by the point-of-contact server by redirecting and/or forwarding user requests to the federated user lifecycle management application as appropriate. Referring again to FIG. **7**, point-of-contact server **702** receives user requests **730**, which are then analyzed to determine the type of request that has been received, which might be indicated by the type of request message that has been received or, as noted above, by determining the destination URI within the request message. While requests **732** for protected resources continue to be forwarded to application servers **704**, requests **734** for federated user lifecycle management functions, e.g., requests to invoke a single-sign-off operation, are forwarded to federated user lifecycle management application **708**, which invokes the appropriate federated user lifecycle management plug-in as necessary to fulfill the received request. When a new federation protocol or a new federated function is defined, or when an existing one is somehow modified or refined, support can be added simply by plugging a new support module or can be refined by modifying a previously installed plug-in.

The exemplary implementation of a federated user lifecycle management application in FIG. **7** illustrates that the federated user lifecycle management application is able to support multiple, simultaneous, federated user lifecycle management functions while providing a pluggability feature, thereby allowing new functionality to be added to the federated user lifecycle management application in the form of a plug-in when needed without requiring any changes to the existing infrastructure. For example, assuming that the present invention is implemented using a Java™-based federated user lifecycle management application, support for a new federation protocol, such as a newly published single-sign-on protocol, can be added by configuring newly developed Java™ classes to the Java™ CLASSPATH of the federated user lifecycle management application, wherein these new classes support the new standard along with the protocol interface for supporting the present invention.

The exemplary federated architecture leverages the existing environment in which a federated user lifecycle management solution is to be integrated. The federated user lifecycle management application can be easily modified to support new protocols/standards as they evolve with minimal changes to the overall infrastructure. Any changes that might be required to support new federated user lifecycle management functionality are located almost exclusively within the federated user lifecycle management application, which would require configuring the federated user lifecycle management application to understand the added functionality.

There may be minimal configuration changes in other federated components, e.g., at a point-of-contact server, in order to allow the overall infrastructure to be able to invoke new federated user lifecycle management functionality while continuing to support existing federated user lifecycle management functionality. However, the federated user lifecycle management applications are functionally independent from the remainder of the federated components in that the federated user lifecycle management applications may require only minimal interaction with other federated components of the federated environment. For example, in an exemplary embodiment, the federated user lifecycle management functionality may integrate with an enterprise-based datastore, e.g., an LDAP datastore, if federated user lifecycle management information, such as NameIdentifier values in accordance with the Liberty Alliance profiles, are to be stored in an externally-accessible federated user lifecycle management datastore as opposed to a private, internal, federated user lifecycle management datastore that is not apparent or accessible to external entities.

Hence, an existing environment needs minimal modifications to support federated user lifecycle management functionality. Moreover, changes to federated user lifecycle management functionality, including the addition of new functionality, have minimal impact on an existing federated environment. Thus, when a new single-sign-on standard is published, support for this standard is easily added.

Traditional user authentication involves interaction between an enterprise's computing environment and the end-user only; the manner in which the enterprise chooses to implement this authentication interchange is the choice of the enterprise, which has no impact on any other enterprise. When federation or cross-domain single-sign-on functionality is desired to be supported, however, it becomes a requirement that enterprise partners interact with each other. This requirement cannot be done scalably using proprietary protocols. Although adding support for standards-based federation protocols directly to a point-of-contact entity seems like a robust solution, the point-of-contact entity, which is already an existing component within the enterprise's computing environment, must be modified; moreover, it must be modified every time that one of these public federation protocols changes. Moving this functionality out of the point-of-contact entity provides a more modular approach, wherein this pluggable functionality makes it easy to maintain migrations or updates to these protocols.

Runtime Linked-User-Account Creation During a Single-Sign-On Operation

With reference now to FIG. **8**, a dataflow diagram depicts a typical prior art HTTP-redirection-based single-sign-on operation that is initiated by a federated identity provider to obtain access to a protected resource at a federated service provider. Although the processes that are illustrated within FIG. **8** and the subsequent figures employ HTTP-based communications, the present invention is not limited to HTTP-based communications, and other communication protocols may be employed; in particular, the present invention is not limited to front-channel communications as represented by HTTP redirection-based techniques but may be equally applied to back-channel techniques, such as SOAP/HTTP or SOAP/MQ. In the dataflow in FIG. **8**, the user of a client, also known as a user agent, such as a web browser application, has

already established a user account not only at the identity provider but also at the service provider (step **802**) and that the user has already authenticated to the identity provider (step **804**).

One of the prerequisites for the exemplary dataflow in FIG. **8** is that, at a minimum, that the user already has a federated account with the service provider; in other words, the service provider is required to recognize user identity information for the user when it receives this information from an identity provider in order to perform a single-sign-on operation for the user when initiated by an identity provider. With step **804**, the prerequisite is simply that the user has authenticated to the identity provider at some previous point in time and currently has a valid session with the identity provider; there are no requirements on the reasons for which a session was established at step **804**. It should be noted that other scenarios are possible, e.g., in which the identity provider determines at some later point in time after some interaction with the user that the identity provider only performs certain operations for authenticated users. It should also be noted that other scenarios for initiating a single-sign-on process are possible; for example, a user could initiate a single-sign-on operation by requesting a protected resource at the service provider, e.g., by using a bookmarked URL within a browser application.

The single-sign-on process commences at the identity provider by sending to the client from the identity provider an offer to provide access to federated resources (step **806**); the offer may be in the form of hyperlinks to resources at federated web sites or, more generally, at federated domains, and the offer may be made in the form of an HTTP response message in response to a previous HTTP request message from the client (not shown) to view a particular web page on the identity provider's web site. The user then selects one of the offered federated resources at service providers that are known to the identity provider (step **808**); the selection may be facilitated by an HTTP request message from the client to the identity provider.

The identity provider builds a message for requesting access to the selected federated resource on behalf of the user such that the message also includes a single-sign-on request (step **810**), and the identity provider sends the resource request with the single-sign-on request to the client (step **812**), e.g., in the form of an HTTP redirect message (HTTP Response message with status/reason code "302"). The redirect message redirects the client to the appropriate location, e.g., as identified by a URI within the "Location" header of the redirect message, that identifies the appropriate service at the service provider that controls access to the requested federated resource.

It should be noted that, in the prior art, a preferred manner of initiating a push-type single-sign-on operation is to include the single sign-on assertion information in a request initiated by the identity provider and sent to the service provider in response to a request triggered directly at the service provider. For example, in a computing environment that implements a single-sign-on operation as specified in the Liberty Alliance ID-FF 1.2 specifications, an AuthnResponse is built. In some scenarios, it is required that the identity provider redirect the client to a trigger URL at the service provider, whereafter the trigger URL will cause the service provider to build a message, e.g., an AuthnRequest message within the Liberty Alliance specifications, that is redirected to the identity provider, which in turn causes the identity provider to build response, e.g., an AuthnResponse; this is the means by which a push-based SSO is implemented with Liberty ID-FF 1.1.

The outgoing request message may comprise two distinct requests: one requested operation for a single-sign-on operation on behalf of the user of the client, and another requested operation for accessing the protected resource that has been selected by the user of the client. Alternatively, the request to access the federated resource may include an implicit request to perform a single-sign-on operation for the user such that the single-sign-on process is merely part of a larger process for accessing the selected resource. The manner in which information is provided for the requested single-sign-on operation may vary. For example, the "Location" HTTP header of the redirect message may also include a query component, e.g., appended to a URI and demarcated within the URI with a "?" character, that contains various information, including security tokens for accomplishing the single-sign-on operation.

In response to receiving the redirect message from the identity provider, the client sends an HTTP Get message to the appropriate service at the service provider as indicated by the URI in the HTTP redirect message from the identity provider (step **814**). In this manner, the client accesses the appropriate service at the service provider because the URI in the HTTP Get message still contains the attached information that is required by the service at the service provider.

The service provider receives the request message from the client, and assuming that the single-sign-on aspect of the request is successfully completed (step **816**) such that the user has an active session at the service provider, the service provider then processes the resource access aspect of the request message (step **818**). After processing the request message, the service provider responds by sending an HTTP response message to the client (step **820**), thereby concluding the process.

With reference now to FIGS. **9**A-**9**B, dataflow diagrams depict an HTTP-redirection-based single-sign-on operation that is initiated by a federated identity provider to obtain access to a protected resource at a federated service provider while performing a runtime linked-user-account creation operation at the federated service provider in accordance with an embodiment of the present invention. In a manner similar to that shown in FIG. **8**, both FIGS. **9**A-**9**B depict a process by which an identity provider may request a single-sign-on operation at a selected service provider. However, whereas FIG. **8** depicts only a generalized single-sign-on operation at step **816**, the processes that are shown in FIGS. **9**A-**9**B differs from the process that is shown in FIG. **8** with respect to the single-sign-on operation. Unlike the process that is shown in FIG. **8**, there is no prerequisite to the processes that are shown in FIGS. **9**A-**9**B that the user of a client has already established a user account at the service provider.

More specifically, prior art solutions for single-sign-on operations have certain prerequisites, namely that the service provider and the identity provider must both recognize the user, and they must have an agreed-upon means of referring to this user, i.e. an alias identifier, or more simply, an alias; both prerequisites must be true before the initiation of a single-sign-on operation in order for the single-sign-on operation to be successful, otherwise it would fail. In these prior art solutions, the alias is included in the single-sign-on response that is sent from the identity provider to the service provider and used by the service provider to identify the user and build a session for the user. These prerequisites exist within prior art solutions whether: (a) the single-sign-on operation is triggered by the service provider, e.g., when a user accesses a protected resource that is hosted at the service provider and the service provider needs a session for the user, thereby initiating a single-sign-on operation by sending a request to an identity provider; or (b) the single-sign-on operation is triggered by the identity provider, e.g., when an identity provider has a list of linked resources that are hosted by related

service providers, and after a user selects one of these links, a single-sign-on operation is initiated by the identity provider. In contrast, the present invention eliminates these prerequisites, as shown by multiple exemplary embodiments of the present invention that are described hereinbelow.

Referring to FIG. **9**A, the user has already authenticated to the identity provider (step **902**). With step **902**, the user has authenticated to the identity provider at some previous point in time and currently has a valid session with the identity provider; there are no requirements on the reasons for which a session was established at step **902**. It should be noted that other scenarios for initiating a single-sign-on operation may have a different sequence of steps. For example, a user might browse information that is provided by an identity provider without an authenticated session; at some later point in time after some interaction with the user, the user requests a resource such that the identity provider determines that the identity provider only performs certain operations for authenticated users, thereafter initiating an authentication operation with the user.

The single-sign-on process commences at the identity provider by sending to the client from the identity provider an offer to provide access to federated resources (step **904**); the offer may be in the form of hyperlinks to resources at federated web sites or, more generally, at federated domains, and the offer may be made in the form of an HTTP response message in response to a previous HTTP request message from the client (not shown) to view a particular web page on the identity provider's web site. The user then selects one of the offered federated resources at service providers that are known to the identity provider (step **906**); the selection may be accomplished by an HTTP request message from the client to the identity provider.

If the user does yet have a federated identity that may be used for a federated single-sign-on operation, then the identity provider creates an alias for the user (step **908**). The identity provider builds a message for requesting access to the selected federated resource on behalf of the user such that the message also includes a single-sign-on request (step **910**), i.e. a push-type single-sign-on request. A push-type single-sign-on request originates with an identity provider and is pushed to a service provider in an unsolicited manner in order to provide the service provider with information that authenticates a user identity; in contrast, a pull-type single-sign-on request originates with a service provider that is attempting to pull authentication information for a user in a solicited manner. Alternatively, a push-type single-sign-on operation can be emulated, particularly when a push-type single-sign-on operation is not supported explicitly. An emulated push-type single-sign-on operation occurs when an identity provider issues a request to a service provider, e.g., via a specialized service at the service provider such as an intersite transfer service, whereupon the service provider issues a single-sign-on request to the identity provider; after the identity provider responds to the request from the service provider, the processing steps are the same as if an explicit push-type single-sign-on operation has occurred.

The identity provider sends the resource request with the single-sign-on request to the client (step **912**), e.g., in the form of an HTTP redirect message (HTTP Response message with status/reason code "302"). The redirect message redirects the client to the appropriate location, e.g., as identified by a URI within the "Location" header of the redirect message, that identifies the appropriate service at the service provider that controls access to the requested federated resource. In response to receiving the redirect message from the identity provider, the client sends an HTTP Get message to the appropriate service at the service provider as indicated by the URI in the HTTP redirect message from the identity provider (step **914**).

The service provider receives and processes the received request message from the client (step **916**). At step **916**, the service provider retrieves from the received message an alias identifier or alias information that has been associated with the user by the identity provider but which is not recognized by the service provider as being associated with a previously existing user account at the service provider. Hence, at step **916**, the service provider determines that the user does not have a user account that links the user with the identity provider, i.e. a linked user account that informs the service provider that the service provider should accept information for a single-sign-on operation from the identity provider in order to authenticate the user. In other words, at step **916**, the service provider determines that the service provider does not have a user account that links a user account at the service provider with a user account at the identity provider.

This recognition is significant for the following reasons. It should be noted that the user may already have one or more accounts at the service provider. These accounts may be used independently because a unique user account is based on a unique user identifier; given a user identifier, the service provider determines the privileges that are associated with the user identifier, i.e. a particular user account. Given that a linked user account is independent of any other user account that the user may have at the service provider, and given that a linked user account requires a unique user identifier to be associated with the linked user account, a linked user account is based on a user identifier that is independent and unique in comparison with any other user identifier that is known to the service provider; this particular user identifier is known as an alias identifier, although some form of alias information within multiple data variables may be used in place of a single alias data variable in some embodiments.

Therefore, after the service provider recognizes that a provided alias identifier is not associated with a previously existing user account at the service provider, the service provider begins to perform operations in order to ensure that the single-sign-on operation is performed successfully. Since the user had not yet been federated with the service provider, the service provider creates a new account for the user with the alias information that has been provided by the identity provider within the request message (step **918**) such that the user has an active session at the service provider.

It should be noted that the minimum information required to create this account will be any local information that is required to identify the account (which is generated internally by the service provider) and any alias information that is provided by the identity provider; thereafter, the newly created user account is linked to the identity provider based on the provided alias for the user such that the service provider is able to perform a single-sign-on operation on behalf of the user in cooperation with the identity provider. If the request to the service provider includes user attributes as well as an alias identifier for the user, then these attributes may be added to the local account; however, it should be noted that, in some embodiments, the service provider may create a linked user account using only the alias identifier without any additional user attribute information from the identity provider, possibly by assigning a local set of default user attributes that are determined by the service provider and that are independent of any information that is received from the identity provider. Whether any additional attributes are provided or not, what should be noted is that this account would preferably not be enabled for direct authentication; hence, the only manner for

the user to gain access to resources at the service provider would be as the result of a session established from a single-sign-on operation triggered by the identity provider.

After the linked user account has been created, the service provider then performs the requested resource access (step **920**). After performing the resource access, the service provider responds by sending an HTTP response message to the client (step **922**), thereby concluding the process.

The single-sign-on operation of the present invention, e.g., the embodiment that is shown in FIG. **9**A, differs from the single-sign-on solutions of the prior art, e.g., the operation that is shown in FIG. **8**, because the service provider recognizes during the single-sign-on operation of the present invention that the service provider does not have a user account for the user that links the user to an account at an identity provider in order to support single-sign-on operations, yet with the present invention the service provider is able to dynamically perform operations to allow the single-sign-on operation to proceed. More specifically, the service provider does not have, e.g., within its user registries or databases, enough information that allows the service provider to determine the user's local identity, and therefore the user's privileges, for the single-sign-on operation; without this information, the service provider cannot determine the appropriate set of privileges to be given to the user and, therefore, the type of session/access control rights to give the user, if the service provider were to allow the single-sign-on operation to proceed. In the prior art, the service provider cannot automatically create an active session for the user and allow access to protected resources; with the present invention, the service provider dynamically performs a runtime linked-user-account creation operation at the service provider by creating a linked user account based on the user identity, and possibly attribute information, that has been provided by the identity provider to the service provider, e.g., as provided in a request that has been redirected from the identity provider through the client. The service provider is willing and able to perform such operations based on its trust relationship with the identity provider with their federated computing environment. In this manner, the single-sign-on request can be fulfilled by the service provider, which results in the creation of an active session for the user, and the request for access to the protected resource can proceed.

In a manner similar to that shown in FIG. **9**A, FIG. **9**B depicts a process by which an identity provider may request a single-sign-on operation at a selected service provider; similar elements in the figures are identified by similar reference numerals. However, whereas FIG. **9**A depicts only a generalized runtime user account creation operation at step **918**, the process that is shown in FIG. **9**B differs from the process that is shown in FIG. **9**A with respect to the runtime linked-user-account creation operation, which stretches over steps **930**-**942** in FIG. **9**B. Unlike the process that is shown in FIG. **9**A, in the process that is shown in FIG. **9**B, the service provider is not able to immediately create, or to complete immediately the creation of, a user account at the service provider based on the information that has been provided by the identity provider to the service provider.

Referring now to FIG. **9**B, the single-sign-on processing in FIG. **9**B (step **930**) differs because the service provider recognizes during the single-sign-on operation that the service provider does not have a pre-existing user account for the identity that is claimed or asserted in the single-sign-on request from the identity provider, i.e. a pre-existing user account that links the service provider to the identity provider on behalf of the user, and further that the information contained in the single-sign-on request is not sufficient to create a valid account at the service provider. Thus, the service provider recognizes that it requires additional information about the user from the identity provider. More specifically, the service provider does not have sufficient user attribute information to create an active account for the user; for example, there may be attributes that are required by the service provider but were not included in the received single-sign-on request. Because of these additional requirements, the service provider is not yet able to create, or to completely create, whatever user account registry/database entries that it requires based on the information it has received.

In the embodiment that is shown in FIG. **9**B, the service provider responds by sending an HTTP redirect message (HTTP Response message with status/reason code "302") to the client (step **932**); the message from the service provider contains a request for additional user attribute information. The redirect message redirects the client to the identity provider using a return URI that was previously provided by the identity provider to the service provider. In response to receiving the redirect message from the service provider, the client sends an HTTP Get message to the identity provider as indicated by the URI in the HTTP redirect message from the service provider (step **934**).

The identity provider then processes the received message (step **936**) to build a response that contains user attribute information that is stored by the identity provider about the user of the client. The manner in which specific user attributes are identified may vary. For example, the user attribute information that is provided by the identity provider may include user attribute information that was explicitly requested by the service provider. Additionally or alternatively, the user attribute information that is provided by the identity provider may include user attribute information that was implicitly requested by the service provider, i.e. user attribute information that has been determined by the identity provider to be sufficient for performing a user account creation operation at the service provider. In addition, the identity provider may perform an operation to reconcile the received request for additional user attribute information with the identity provider's previous single-sign-on request to ensure that a service provider is not attempting to obtain user attributes without sufficient reason to do so.

After the identity provider builds the message at step **936**, the identity provider sends the message with the additional user attributes to the client (step **938**), e.g., in the form of an HTTP redirect message (HTTP Response message with status/reason code "302"). The redirect message redirects the client to the appropriate location, e.g., as identified by a URI within the "Location" header of the redirect message, that identifies the appropriate service at the service provider; the appropriate location may have been provided by the service provider in its request for the additional user attributes. In response to receiving the redirect message from the identity provider, the client sends an HTTP Get message to the appropriate service at the service provider as indicated by the URI in the HTTP redirect message from the identity provider (step **940**).

Given the additional user attribute information in the received message, the service provider performs a runtime linked-user-account creation operation at the service provider (step **942**) by creating a linked user account based on the received user attribute information; in this manner, the single-sign-on request (and any subsequent single-sign-on requests) can be fulfilled by the service provider, which also creates an active session for the user, and the request for access to the protected resource can proceed at steps **920** and **924**. In some embodiments, the service provider may preliminarily create

the user account yet postpone the completion of the creation of the user account; in these embodiments, the service provider completes the creation of the user account at step **942**. It should be noted that the processes are described in the context of an HTTP redirection-based attribute retrieval, but these processes could be implemented with a back-channel approach, such as a SOAP/HTTP-based SAML attribute query from the service provider to the identity provider.

With reference now to FIGS. **9**C-**9**E, dataflow diagrams depict an HTTP-redirection-based single-sign-on operation that is initiated by a federated identity provider to obtain access to a protected resource at a federated service provider with alternative methods for obtaining user attributes by the federated service provider in accordance with an embodiment of the present invention. The processes that are shown in FIGs. FIGS. **9**C-**9**E differ from the processes that are shown in FIGS. **9**A-**9**B primarily in the manner in which the service provider obtains user attributes during a single-sign-on operation; the processes that are shown in FIGS. **9**A-**9**B and in FIGS. **9**C-**9**E are both initiated by an identity provider. In addition, in the dataflow that is shown in FIG. **9**B, a service provider performs the runtime linked-user-account creation operation at step **942**; in contrast, in the dataflow that is shown in FIGS. **9**C-**9**E, the service provider may perform the runtime linked-user-account creation operation over multiple steps by partially creating a linked user account and then later completing the runtime linked-user-account creation operation, as described in more detail hereinbelow.

Referring now to FIG. **9**C, the process commences with a user browsing public resources that are hosted by an identity provider (step **952**), i.e. resources that are not protected resources that require an authentication operation with respect to the user prior to accessing the resources. At some subsequent point in time, the user's client sends to the identity provider a request for access to a protected resource that requires an authentication operation (step **954**); for example, the user might attempt to browse information that the identity provider maintains about resources at service providers within a federated computing environment in which the identity provider participates with service providers, i.e. the identity provider's federated partners. In response to the user request, the identity provider performs an authentication operation with the user (step **956**). In this example, the identity provider thereafter offers links to resources at federated service providers (step **958**), and the user then selects or initiates an operation to access a resource at a service provider (step **960**).

If the user does yet have an alias identifier that may be used for a federated single-sign-on operation, then the identity provider creates an alias for the user (step **962**), which may include performing other operations to associate the alias identifier with the user, particular to associate the alias identifier with other information that is associated with the user, e.g., user attribute information. The identity provider builds a message for requesting access to the selected federated resource on behalf of the user such that the message also includes a single-sign-on request (step **964**), i.e. a push-type single-sign-on request. The identity provider sends the resource request with the single-sign-on request to the client (step **966**), e.g., in the form of an HTTP redirect message (HTTP Response message with status/reason code "302"). The redirect message redirects the client to the appropriate location, e.g., as identified by a URI within the "Location" header of the redirect message, that identifies the appropriate service at the service provider that controls access to the requested federated resource. In response to receiving the redirect message from the identity provider, the client sends an HTTP Get message to the appropriate service at the service provider as indicated by the URI in the HTTP redirect message from the identity provider (step **968**).

The service provider then processes the single-sign-on response (step **970**), which may include, e.g., extracting the newly created alias identifier for the user and extracting any user attribute information about the user that has been preliminarily provided by the identity provider to the service provider. The service provider attempts to create a new user account (step **972**) but either is not able to immediately create or is not able to create a fully appropriate user account at the service provider based on the federated user identity information that has been provided by the identity provider to the service provider in the single-sign-on response. In other words, depending on the implementation, the service provider either fails to create or determines that it cannot create a user account at this point in time, or the service provider creates a user account with limited-time or limited-access privileges. More specifically, the service provider does not have, e.g., within the combination of its user registries or databases and the information that is initially provided by the identity provider, the information required to allow the service provider to determine the user's appropriate set of privileges; without this information, the service provider cannot determine the type of session/access control rights to give the user. Hence, the service provider recognizes during the single-sign-on operation that the service provider does not yet have a linked user account for the user while also recognizing that the service provider requires additional information about the user from the identity provider (step **974**).

The manner in which the service provider pulls additional user attribute information from the identity provider to complete the user account creation operation may vary; two examples of variations are shown in FIG. **9**D and in FIG. **9**E. FIG. **9**D illustrates a front-channel attribute retrieval operation, whereas FIG. **9**E illustrates a back-channel attribute retrieval operation. Hence, the data flow diagram that is shown in FIG. **9**C continues into either FIG. **9**D or FIG. **9**E; in both cases, the dataflow diagrams in FIG. **9**D and FIG. **9**E conclude with similar steps that are denoted with similar reference numerals.

Referring now to FIG. **9**D, the service provider responds to its determination of its lack of user attribute information by sending an HTTP redirect message (HTTP Response message with status/reason code "302") to the client (step **976**); the message from the service provider contains a request for additional user attribute information. The redirect message redirects the client to the identity provider using a return URI that was previously provided by the identity provider to the service provider. In response to receiving the redirect message from the service provider, the client sends an HTTP Get message to the identity provider as indicated by the URI in the HTTP redirect message from the service provider (step **978**). The identity provider then processes the received message (step **980**) to build a response that contains user attribute information that is stored by the identity provider about the user of the client. The manner in which specific user attributes are identified may vary. For example, the user attribute information that is provided by the identity provider may include user attribute information that was explicitly requested by the service provider. Additionally or alternatively, the user attribute information that is provided by the identity provider may include user attribute information that was implicitly requested by the service provider, i.e. user attribute information that has been determined by the identity provider to be sufficient for performing a user account creation operation at the service provider. In addition, the identity provider may

perform an operation to reconcile the received request for additional user attribute information with the service provider's previous single-sign-on request to ensure that a service provider is not attempting to obtain user attributes without sufficient reason to do so.

After the identity provider builds the message at step **980**, the identity provider sends the message with the user attributes to the client (step **982**), e.g., in the form of an HTTP redirect message (HTTP Response message with status/reason code "302"). In response to receiving the redirect message from the identity provider, the client sends an HTTP Get message to the appropriate service at the service provider as indicated by the URI in the HTTP redirect message from the identity provider (step **984**).

Given the user attribute information in the received message, the service provider performs or completes a user account creation operation at the service provider (step **986**) based on the received user attribute information; in this manner, the single-sign-on request can be fulfilled by the service provider, which also creates an active session for the user, and the request for access to the protected resource can proceed. In both FIG. **9**D and FIG. **9**E, the service provider provides access to the originally requested protected resource (step **988**), and the service provider sends an HTTP Response message to the client that contains information that is derived from accessing the protected resource (step **990**), thereby concluding the process.

Referring now to FIG. **9**E, the service provider responds to its determination of its lack of user attribute information by sending a SOAP message directly from the service provider to the identity provider (step **992**); the message from the service provider contains a request for required user attribute information. The identity provider then processes the received message (step **994**) to build a response that contains user attribute information that is stored by the identity provider about the user of the client; again, the manner in which specific user attributes are identified may vary as described above. The identity provider sends a SOAP response message with the required user attributes to the client (step **996**). Given the user attribute information in the received message, the service provider performs or completes a runtime user account creation operation at the service provider (step **998**) based on the received user attribute information; in this manner, the single-sign-on request can be fulfilled by the service provider, which also creates an active session for the user, and the request for access to the protected resource can proceed at steps **988** and **990**.

With reference now to FIG. **10**, a flowchart depicts a more detailed process for performing a runtime linked-user-account creation operation at a service provider during a single-sign-on operation that has been initiated by an identity provider. The flowchart that is shown in FIG. **10** depicts a service-provider-centered perspective for some of the processing that occurs at a service provider within the dataflow diagram that is shown in FIG. **9**B.

The process commences when the service provider receives a request from an identity provider to access a protected resource at the service provider on behalf of a user of a client based on a single-sign-on operation (step **1002**). It should be noted that the protected resource may be an endpoint that corresponds to the service provider's functionality for fulfilling a single-sign-on request; in other words, it is possible that the request from the identity provider is redirected directly to the known functionality to accomplish the single-sign-on operation because the user has not requested a particular back-end resource but simply overall access to the service provider. The service provider extracts a user identifier from the received request message (step **1004**) and makes a determination as to whether or not the user identifier is recognized (step **1006**).

If the user identifier is not recognized by the service provider, then the service provider cannot create an active session with the appropriate security considerations for protected access to its hosted computational resources. The service provider extracts any user attribute information that might be embedded in the received message (step **1008**), and a determination is made as to whether or not the service provider has sufficient information about the user to create a user account for the user at that time (step **1010**), e.g., as would be required to generate an entry into a user registry or whatever operations are typically performed by the service provider to create a local user account for the user at the service provider.

If the service provider does not have sufficient information about the user to create a user account for the user, the service provider may send a request to the identity provider to obtain additional user attribute information (step **1012**); this may be performed in a synchronous or asynchronous manner. The service provider subsequently receives additional user attributes from the identity provider (step **1014**).

It should be noted that an embodiment of the present invention may be implemented such that a user attribute retrieval operation by a service provider may be performed with respect to an attribute information provider. For example, a service provider may send a request for user attributes to an attribute information provider rather than an identity provider. Alternatively, a service provider may send a request for user attributes to an identity provider, which subsequently enlists assistance by sending a request to an attribute information provider, thereby acting as an intermediate trusted agent on behalf of the service provider. Additional information about the usage of an attribute information provider within the context of a federated computing environment may be found in Blakley et al., "Method and system for user-determined attribute storage in a federated environment", U.S. Patent Application Publication US 2004/0128378 A1, published Jul. 1, 2004, based on U.S. patent application Ser. No. 10/334,605, filed Dec. 31, 2002, which has a common assignee with the present patent application and is hereby incorporated by reference.

After receiving the additional user attributes at step **1014**, or based on any user attribute information that may have been within the original request as extracted at step **1008**, the service provider performs a linked user account creation operation (step **1016**); as noted above, the operations that are performed to create a linked user account for the user at the service provider may vary. After the linked user account has been created at the service provider, the service provider may be able to proceed with the processing of the original request.

The service provider then determines whether there is sufficient information for activating the newly created account for the user (step **1018**). If the service provider does not yet have sufficient information for activating the user's account, then a determination is made as to whether the service provider has already made too many attempts to obtain user attributes for activating the user account (step **1020**); if not, then the process branches back to step **1012** to make an attempt to obtain the necessary user attributes. Otherwise, the service provider performs some type of error handling (step **1022**), which may entail sending an error response back to the identity provider.

If the service provider has sufficient information for activating the user's account at step **1018**, then the service provider creates an active session for the user (step **1024**), either based on a successful authentication of a recognized user

identity at step **1006** or based on the newly created user identity at step **1016**. The service provider then generates a response for the original request to access the protected resource (step **1026**), and the response is sent by the service provider to the identity provider (step **1028**), thereby concluding the process.

The exemplary dataflow that is described with respect to FIG. **9**B or the exemplary process that is described with respect to FIG. **10** can be described in the context of the exemplary data processing systems that are shown in FIG. **7** or in FIG. **3**.

In order to perform a single-sign-on operation, the identity and attribute service (I&AS) **356** or **718** needs to be able to recognize a user identity from a single-sign-on request within a user registry in some manner. The user registry can be a local registry, e.g., private to the installation of the federated functionality, such as federation user registry **358** or **720**, or it could be an enterprise registry that is shared by other applications within an enterprise, such as enterprise user registry **338** or **722**. The user account information or user registry information needs to allow I&AS **356** or **718** to build the appropriate information that identifies the user locally. This information may be represented by a username, a set of attributes, e.g., groups and/or roles and/or entitlements, or an opaque identifier, e.g., a NameIdentifier as described within the Liberty Alliance specifications. If the information asserted by the identity provider about the user for which a single-sign-on is being requested cannot be found within a local registry, then I&AS **356** or **718** is not able to build information about the user in a local manner, e.g., to create valid credentials that are used by entities within the local enterprise; in addition, the point-of-contact server **342** or **702** is not able establish a session for the user, and the user is not able to access protected resources.

The present invention provides a mechanism to prevent the generation of some form of error code for an unrecognized user as would be performed in prior art approaches. In the embodiment of the present invention that is shown in the figures, I&AS **356** or **718** may attempt to create a user account, record, or entry, as appropriate for the data storage requirements of the enterprise when the received user identity information is unrecognized. I&AS **356** or **718** performs these operations based on the trust relationship between the federated partners of the identity provider and the service provider, e.g., as provided for within a configured policy that allows for this type of action. The user identity information that is received in the original request may be an actual username value or an opaque data value such as a Liberty Alliance NameIdentifier; this user identity information may be used as a pointer into a user registry from which this user's information would be accessed at some subsequent point in time, or it may be used in a temporary manner until permanent data records are created, e.g., as a pointer into a cache that contains temporary information.

If the original single-sign-on request contains attribute data about a user, then the attribute information may be added directly to a user registry when creating a user account for the user locally. However, the originally received request does not necessarily contain all of the information that is required locally to create a user account for the user. In this case, I&AS **356** or **718** working with single-sign-on protocol service (SPS) **354** or **716** may issue an attribute query, such as a SAML attribute query and/or a WS-AttributeService request, to the user's identity provider to retrieve additional information about the user. As a result, the service provider is able to create a user account for the user in a runtime manner, i.e. while the single-sign-on operation is suspended or, from another temporal viewpoint, concurrently with or as part of the single-sign-on operation.

The level of trust that is accorded to this type of runtime linked-user-account creation operation by the service provider may vary; it is not necessarily the case that all service providers would be willing to allow a completely automated, dynamically determined, linked-user-account creation operation. Hence, in some cases, a service provider may require that a local workflow operation must be undertaken, e.g., a type of local administrative approval process. Rather than directly creating a user account for a user in a completely automated process, the federated user lifecycle management (FULM) application/service **352** or **708** might store the user information, including additional, out-of-band-retrieved user attributes, into some form of local datastore, which may, in turn, trigger a local workflow/approval process. This requirement allows a administrative user at the service provider to approve the creation of the user account in accordance with local policy requirements, etc. As a result, the runtime user account creation operation may be asynchronous, in which case the service provider may send a message to the user to indicate that an account is being created for the user at the service provider and to indicate that the user can attempt to perform a login operation at the service provider at some subsequent point in time. If the user needs to be added to more than an I&AS-accessible user registry, then FULM service **352** or **708** may send this information to a local datastore, from which a local user account creation process is initiated; in this case, accounts/records can be created for the user in multiple destinations in addition to a local I&AS-accessible datastore. It should be noted that the service provider may choose to grant the user a limited-time, limited-access session, thereby restricting the user to a subset of accessible resources rather than a complete set of resources until some later event, e.g., completion of some type of workflow process for approving broader access, or until some later determination is made as to the privileges that should be given to the user; thereafter, the user account would be created with an appropriate level of access to resources.

With reference now to FIG. **11**A, a dataflow diagram depicts an HTTP-redirection-based pull-type single-sign-on operation that is initiated by a federated service provider to allow access to a protected resource at the federated service provider while performing a runtime linked-user-account creation operation at the federated service provider in accordance with an embodiment of the present invention. The processes that are shown in FIGS. **11**A-**11**D differ from the processes that are shown in FIGS. **9**A-**9**B primarily in the perspective of the origination of the single-sign-on operation; the processes that are shown in FIGS. **9**A-**9**B are initiated by an identity provider, whereas the processes that are shown in FIGS. **11**A-**11**D are initiated by a service provider.

Referring now to FIG. **11**A, the process commences with a user browsing public resources that are hosted by a service provider (step **1102**), i.e. resources that are not protected resources that require an authentication operation with respect to the user prior to accessing the resources. At some subsequent point in time, the user's client sends to the service provider a request for access to a protected resource that requires an authentication operation (step **1104**). The service provider recognizes that it does not have an identifier for the user's preferred identity provider, and the service provider prompts the user to provide the information in some manner (step **1106**), e.g., through user-selectable controls in a web page. The client then sends to the service provider the appro-

priate information about the preferred identity provider as selected by the user (step **1108**).

The service provider then builds a pull-type single-sign-on request for the user (step **1110**). In this example, the service provider assumes that the user is a federated user, e.g., by assuming that the identity provider maintains some type of federated alias identifier for the user. The service provider sends the single-sign-on request to the client (step **1112**), e.g., in the form of an HTTP redirect message (HTTP Response message with status/reason code "302"). In response to receiving the redirect message from the service provider, the client sends an HTTP Get message to the identity provider as indicated by the URI in the HTTP redirect message from the service provider (step **1114**).

In response to receiving the pull-type single-sign-on request, the identity provider performs an authentication operation with the client with respect to the user of the client, if required (step **1116**); for example, the authentication operation may not be required if the user is already logged on at the identity provider, i.e. if the user already has an active session at the identity provider. The identity provider evaluates the received request (step **1118**), and the identity provider determines that the service provider has not provided a user identity to the identity provider as part of the federation request. If the identity provider further determines that the user does not yet have a federated identity, then the identity provider creates a federated identity for the user, e.g., an alias identifier for the user (step **1120**). The identity provider builds a pull-type single-sign-on response (step **1122**) that contains the newly created, federated identity for the user and optionally contains additional user attribute information that is managed by the identity provider about the user; this pull-type single-sign-on response may or may not have the same data format characteristics as a push-type single-sign-on response. The identity provider sends the single-sign-on response to the client (step **1124**), e.g., in the form of an HTTP redirect message (HTTP Response message with status/reason code "302"). In response to receiving the redirect message from the identity provider, the client sends an HTTP Get message to the service provider as indicated by the URI in the HTTP redirect message from the identity provider (step **1126**).

The service provider then processes the single-sign-on response (step **1128**), which may include, e.g., extracting the newly created federated identifier for the user and may also include the extraction of additional user attribute information about the user. The service provider then creates a new linked user account for the user with the federated identifier that was provided by the identity provider (step **1130**), which may possibly also entail using some user attribute information about the user from the identity provider if the identity provider has sent such information and if the service provider needs this information while creating the new user account. After the user has an active session based on the newly created user account, the service provider provides access to the originally requested protected resource (step **1132**), and the service provider sends an HTTP Response message to the client that contains information that is derived from accessing the protected resource (step **1134**), thereby concluding the process.

With reference now to FIGS. **11**B-**11**D, a set of dataflow diagrams depict an HTTP-redirection-based pull-type single-sign-on operation that is initiated by a federated service provider to allow access to a protected resource at the federated service provider with additional retrieval of user attribute information from a federated identity provider while performing a runtime linked-user-account creation operation at the federated service provider in accordance with an embodiment of the present invention. In a manner similar to that shown in FIG. **11**A, FIG. **11**B depicts a process by which a service provider may request a single-sign-on operation at a selected identity provider; similar elements in the figures are identified by similar reference numerals. However, whereas FIG. **11**A depicts only a generalized runtime linked-user-account creation operation at step **1130**, the process that is shown in FIG. **11**B differs from the process that is shown in FIG. **11**A with respect to the runtime linked-user-account creation operation, which stretches over steps **1150**-**1164** in FIGS. **11**A-**11**B.

Referring now to FIG. **11**B, unlike the process that is shown in FIG. **11**A, the service provider attempts to create a new user account (step **1150**) but either is not able to immediately create or is not able to create a fully appropriate user account at the service provider based on the federated user identity information that has been provided by the identity provider to the service provider in the single-sign-on. In other words, depending on the implementation, the service provider either fails to create or determines that it cannot create a user account at this point in time, or the service provider creates a user account with limited-time or limited-access privileges. Hence, the single-sign-on processing in FIG. **11**B differs because the service provider recognizes during the single-sign-on operation that the service provider does not yet have a linked user account for the user while also recognizing that the service provider requires additional information about the user from the identity provider (step **1152**). More specifically, the service provider does not have, e.g., within its user registries or databases, enough information that allows the service provider to determine the user's appropriate set of privileges; without this information, the service provider cannot determine the type of session/access control rights to give the user.

The manner in which the service provider pulls additional user attribute information from the identity provider to complete the linked-user-account creation operation may vary; two examples of variations are shown over steps **1154**-**1164** in FIG. **11**C or over steps **1172**-**1178** in FIG. **11**D. FIG. **11**C illustrates a front-channel attribute retrieval operation, whereas FIG. **11**D illustrates a back-channel attribute retrieval operation. Hence, the data flow diagram that is shown in FIG. **11**B continues into either FIG. **11**C or FIG. **11**D; in both cases, the dataflow diagrams in FIG. **11**C and FIG. **11**D conclude with similar steps. In both FIG. **11**C and FIG. **11**D, the service provider provides access to the originally requested protected resource at step **1132**, and the service provider sends an HTTP Response message to the client that contains information that is derived from accessing the protected resource at step **1134**, thereby concluding the process.

Referring now to FIG. **1**C, the service provider responds to its determination of its lack of user attribute information by sending an HTTP redirect message (HTTP Response message with status/reason code "302") to the client (step **1154**); the message from the service provider contains a request for additional user attribute information. The redirect message redirects the client to the identity provider using a return URI that was previously provided by the identity provider to the service provider. In response to receiving the redirect message from the service provider, the client sends an HTTP Get message to the identity provider as indicated by the URI in the HTTP redirect message from the service provider (step **1156**). The identity provider then processes the received message (step **1158**) to build a response that contains user attribute information that is stored by the identity provider about the user of the client. The manner in which specific user

attributes are identified may vary. For example, the user attribute information that is provided by the identity provider may include user attribute information that was explicitly requested by the service provider. Additionally or alternatively, the user attribute information that is provided by the identity provider may include user attribute information that was implicitly requested by the service provider, i.e. user attribute information that has been determined by the identity provider to be sufficient for performing a linked-user-account creation operation at the service provider. In addition, the identity provider may perform an operation to reconcile the received request for additional user attribute information with the service provider's previous single-sign-on request to ensure that a service provider is not attempting to obtain user attributes without sufficient reason to do so.

After the identity provider builds the message at step **1158**, the identity provider sends the message with the additional user attributes to the client (step **1160**), e.g., in the form of an HTTP redirect message (HTTP Response message with status/reason code "302"). In response to receiving the redirect message from the identity provider, the client sends an HTTP Get message to the appropriate service at the service provider as indicated by the URI in the HTTP redirect message from the identity provider (step **1162**).

Given the additional user attribute information in the received message, the service provider performs or completes a runtime linked-user-account creation operation at the service provider (step **1164**) based on the received user attribute information; in this manner, the single-sign-on request can be fulfilled by the service provider, which also creates an active session for the user, and the request for access to the protected resource can proceed at steps **1132** and **1134**.

Referring now to FIG. **11**D, the service provider responds to its determination of its lack of user attribute information by sending a SOAP message directly from the service provider to the identity provider (step **1172**); the message from the service provider contains a request for additional user attribute information. The identity provider then processes the received message (step **1174**) to build a response that contains user attribute information that is stored by the identity provider about the user of the client; again, the manner in which specific user attributes are identified may vary as described above. The identity provider sends a SOAP response message with the additional user attributes to the client (step **1176**). Given the additional user attribute information in the received message, the service provider performs or completes a runtime linked-user-account creation operation at the service provider (step **1178**) based on the received user attribute information; in this manner, the single-sign-on request can be fulfilled by the service provider, which also creates an active session for the user, and the request for access to the protected resource can proceed at steps **1132** and **1134**.

CONCLUSION

The advantages of the present invention should be apparent in view of the detailed description of the invention that is provided above. When an identity provider attempts to initiate a single-sign-on operation on behalf of a user at a service provider in order to obtain access to a controlled resource that is hosted by the service provider, it is possible that the service provider would not recognize the user identifier or other user identity information in the received request. In the prior art, this scenario would generate an error.

With the present invention, the service provider may dynamically perform a linked-user-account creation operation as part of the single-sign-on operation. If necessary, the service provider can pull additional user attribute information from the identity provider in order to obtain the required user attributes for the user in a manner that is locally required by the identity management functionality of the service provider.

"It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes associated with the present invention are capable of being distributed in the form of instructions in a computer readable medium. Examples of computer readable media include media such as EPROM, ROM, tape, paper, floppy disc, hard disk drive, RAM, and CD-ROMs."

A method is generally conceived to be a self-consistent sequence of steps leading to a desired result. These steps require physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It is convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, parameters, items, elements, objects, symbols, characters, terms, numbers, or the like. It should be noted, however, that all of these terms and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

The description of the present invention has been presented for purposes of illustration but is not intended to be exhaustive or limited to the disclosed embodiments. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiments were chosen to explain the principles of the invention and its practical applications and to enable others of ordinary skill in the art to understand the invention in order to implement various embodiments with various modifications as might be suited to other contemplated uses.

What is claimed is:

1. A method for managing user authentication within a distributed data processing system, wherein a first system and a second system interact within a federated computing environment and support single-sign-on operations in order to provide access to protected resources, at least one of the first system and the second system comprising a processor, the method comprising;

triggering a single-sign-on operation on behalf of the user in order to obtain access to a protected resource that is hosted by the second system, wherein the second system requires a user account for the user to complete the single-sign-on operation prior to providing access to the protected resource;

receiving from the first system at the second system an identifier associated with the user; and

creating a user account for the user at the second system based at least in part on the received identifier associated with the user after triggering the single-sign-on operation but before generating at the second system a response for accessing the protected resource, wherein the created user account supports single-sign-on operations between the first system and the second system on behalf of the user.

2. The method of claim **1** further comprising:

creating an alias identifier for the user at the first system after triggering the single-sign-on operation.

3. The method of claim **1** further comprising:

sending a message from the second system to the first system to pull authentication information for the user

from the first system to the second system in order to trigger the single-sign-on operation for the user at the second system.

4. The method of claim **1** further comprising:

receiving a message from the first system at the second system to push authentication information for the user from the first system to the second system in order to trigger the single-sign-on operation for the user at the second system.

5. The method of claim **1** further comprising:

in response to a determination at the second system that the second system does not have sufficient user attribute information to complete creation of a user account for the user at the second system, sending a request message from the second system to the first system to retrieve user attribute information; and

receiving at the second system from the first system a response message that contains user attribute information that is employed by the second system to complete creation of a user account for the user at the second system.

6. The method of claim **5** further comprising:

employing a front-channel information retrieval mechanism to obtain user attribute information.

7. The method of claim **6** further comprising:

using HyperText Transport Protocol (HTTP) in the front-channel information retrieval mechanism.

8. The method of claim **5** further comprising:

employing a back-channel information retrieval mechanism to obtain user attribute information.

9. The method of claim **8** further comprising:

using Simple Object Access Protocol (SOAP) in the back-channel information retrieval mechanism.

10. The method of claim **5** further comprising:

performing a preliminary user account creation operation to commence creation of the user account for the user prior to retrieving user attribute information for the user; and

performing a concluding user account creation operation to complete creation of the user account for the user after retrieving user attribute information for the user.

11. The method of claim **5** further comprising:

retrieving user attribute information from a fourth system by the first system.

12. The method of claim **1** wherein the first system supports an identity provider and the second system supports a service provider.

13. The method of claim **12** further comprising:

prompting the user by the service provider to provide or to select an identifier for the identity provider prior to receiving an identifier associated with the user.

14. The method of claim **1** further comprising:

in response to a determination at the second system that the second system does not have sufficient user attribute information to complete creation of a user account for the user at the second system, sending a request message to a fourth system to retrieve user attribute information; and

receiving at the second system from the fourth system a response message that contains user attribute information that is employed by the second system to complete creation of a user account for the user at the second system.

15. A computer program product on a computer-readable medium for managing user authentication within a data processing system, wherein a first system and a second system interact within a federated computing environment and support single-sign-on operations in order to provide access to protected resources, at least one of the first system and the second system comprising a processor, the computer program product holding computer program instructions which when executed by the data processing system perform a method comprising:

triggering a single-sign-on operation on behalf of the user in order to obtain access to a protected resource that is hosted by the second system, wherein the second system requires a user account for the user to complete the single-sign-on operation prior to providing access to the protected resource;

receiving from the first system at the second system an identifier associated with the user; and

creating a user account for the user at the second system based at least in part on the received identifier associated with the user after triggering the single-sign-on operation but before generating at the second system a response for accessing the protected resource, wherein the created user account supports single-sign-on operations between the first system and the second system on behalf of the user.

16. The computer program product of claim **15** wherein the method further comprises:

creating an alias identifier for the user at the first system after triggering the single-sign-on operation.

17. The computer program product of claim **15** wherein the method further comprises:

sending a request message from the second system to the first system to retrieve user attribute information in response to a determination at the second system that the second system does not have sufficient user attribute information to complete creation of a user account for the user at the second system; and

receiving at the second system from the first system a response message that contains user attribute information that is employed by the second system to complete creation of a user account for the user at the second system.

18. An apparatus for managing user authentication within a data processing system, wherein a first system and a second system interact within a federated computing environment and support single-sign-on operations in order to provide access to protected resources, at least one of the first system and the second system comprising a processor, the apparatus comprising:

a processor;

a computer memory holding computer program instructions which when executed by the processor perform a method comprising:

triggering a single-sign-on operation on behalf of the user in order to obtain access to a protected resource that is hosted by the second system, wherein the second system requires a user account for the user to complete the single-sign-on operation prior to providing access to the protected resource;

receiving from the first system at the second system an identifier associated with the user; and

creating a user account for the user at the second system based at least in part on the received identifier associated with the user after triggering the single-sign-on operation but before generating at the second system a response for accessing the protected resource, wherein the created user account supports single-sign-on operations between the first system and the second system on behalf of the user.

19. The apparatus of claim **18** wherein the method further comprises:

creating an alias identifier for the user at the first system after triggering the single-sign-on operation.

20. The apparatus of claim **18** wherein the method further comprises:

sending a request message from the second system to the first system to retrieve user attribute information in response to a determination at the second system that the second system does not have sufficient user attribute information to complete creation of a user account for the user at the second system; and

receiving at the second system from the first system a response message that contains user attribute information that is employed by the second system to complete creation of a user account for the user at the second system.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 7,631,346 B2
APPLICATION NO. : 11/097587
DATED : December 8, 2009
INVENTOR(S) : Hinton et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

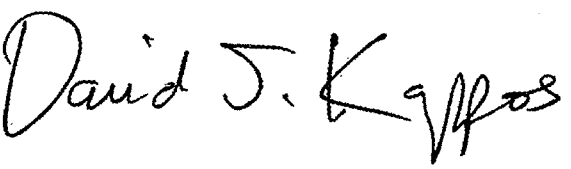
On the Title Page:

The first or sole Notice should read --

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1217 days.

Signed and Sealed this

Twenty-first Day of December, 2010



David J. Kappos
Director of the United States Patent and Trademark Office